Breves Notas sobre
Autómatas y Lenguajes

Jorge L. Ortega Arjona
Departamento de Matemáticas
Facultad de Ciencias, UNAM

Octubre 2004

Índice general

Prefacio

Las *Breves Notas sobre Autómatas y Lenguajes* introducen en forma simple y sencilla a algunos de los temas relevantes en el área de Autómatas y Lenguajes. No tiene la intención de substituir a los diversos libros y publicaciones formales en el área, ni cubrir por completo los cursos relacionados, sino más bien, su objetivo es exponer brevemente y guiar al estudiante a través de los temas que por su relevancia se consideran esenciales para el conocimiento básico de esta área, desde una perspectiva del estudio de la Computación.

Los temas principales que se incluyen en estas notas son: Autómata Finito, la Jerarquía de Chomsky, Lenguajes Regulares, Gramáticas Generativas y Autómatas Celulares. Estos temas se exponen haciendo énfasis en los elementos que el estudiante (particularmente el estudiante de Computación) debe aprender en las asignaturas que se imparten como parte de la Licenciatura en Ciencias de la Computación, Facultad de Ciencias, UNAM.

Jorge L. Ortega Arjona
Octubre 2004

Capítulo 1

Autómata Finito
La Caja Negra

Ocasionalmente, uno se encuentra con algún dispositivo cuya función precisa es incierta o desconocida. En general, si su apariencia no da idea alguna de cuál es su función, normalmente se nombra a este tipo de dispositivos como *caja negra*. Una forma de descubrir cómo trabaja el dispositivo es desarmarlo pieza por pieza, y deducir su función mediante análisis de sus componentes e interconexiones. Esto, sin embargo, no siempre es posible ni necesario. Por otro lado, dado que el dispositivo misterioso normalmente presenta partes tanto de entrada como de salida, es posible descubrir qué hace sin tener que desarmarlo.

Imagínese una caja negra particular (figura 1.1) que presenta una terminal eléctrica marcada como "ENTRADA", un pequeño foco etiquetado "ACEPTADO", y un botón marcado como "REINICIA".

Figura 1.1: Una caja negra

El dispositivo parece aceptar dos tipos de voltaje, y representando estos con los símbolos 0 y 1, se pueden realizar algunos experimentos dando

secuencias de ceros y unos a la caja negra. Al principio, aplicar algunas secuencias parece no afectar al dispositivo. Después de varios intentos, el foco repentinamente se enciende: la secuencia ha sido "aceptada". Tras registrar la secuencia de símbolos que encendió el foco, se repite el experimento, pero la secuencia falla y el foco ya no enciende. En el desconcierto, es posible comenzar a considerar que el circuito interno de la caja se comporta en forma aleatoria. Entonces, alguien más intenta la misma secuencia, pero tras haber presionado el botón de reinicio. El foco se enciende de nuevo. Continuando con los experimentos se descubren varias otras secuencias que, tras haber presionado el botón de reinicio, también encienden el foco. Notando esto, es interesante preguntarse: ¿hasta dónde puede llegarse analizando el comportamiento de la caja negra?

Para hacer tal análisis, es necesario suponer que el contenido de la caja se encuentra en alguna clase de equilibrio estable entre una entrada y otra. También se supone que el circuito interior de la caja puede encontrarse en sólo un número finito de tales estados de equilibrio. Estas suposiciones son perfectamente razonables, al menos si se trata de un dispositivo manufacturado.

Cuando se describe de la forma anterior, la caja negra es esencialmente un *autómata finito*. En términos abstractos, este dispositivo consiste de:

- Una colección finita de Q estados (por ejemplo, las condiciones electrónicas internas y estables del circuito).
- Un alfabeto finito Σ de señales de entrada.
- Una función δ la cual, para toda posible combinación del estado actual y la entrada, determina un nuevo estado.

Dos de los Q estados son especiales: uno llamado *estado inicial*, y otro que es un *estado de aceptación* (a veces llamado también *estado final*).

De esta forma nótese que el botón "REINICIA" coloca al autómata finito en el interior de la caja negra en un estado inicial. Algunas secuencias de ceros y unos dan como resultado que el autómata llegue al estado de aceptación, lo que causa que el foco se encienda.

Se dice que un autómata finito *acepta* cualquier secuencia de símbolos si lo colocan en alguno de sus estados finales. El conjunto de tales secuencias puede verse como las palabras en el *lenguaje* del autómata. Con esta terminología, entonces, se puede preguntar: ¿es posible determinar el lenguaje

del autómata finito dentro de la caja negra? ¿se puede describir brevemente el conjunto de todas las secuencias binarias que causan que el foco se prenda? Al fin y al cabo, éstas no son mas que *palabras* del lenguaje. Ya que el autómata finito dentro de la caja negra sólo tiene un número finito de estados, se puede suponer que esto es un objetivo razonable. Sin embargo, tras un número de experimentos, surgen varias dudas. Por ejemplo, la caja negra acepta las siguientes palabras:

0101
0100101
0100100101
0100100100101

No toma mucho tiempo en darse cuenta que el autómata acepta todas las palabras que tienen la forma $01(001) * 01$, donde $(001)*$ es sólo una abreviación de "todo número de repeticiones de la secuencia 001". De hecho, podría intentarse dibujar parte del *diagrama de transición de estados* del autómata (figura 1.2), en el cual los estados se representan por círculos etiquetados, y las transiciones entre ellos como flechas.

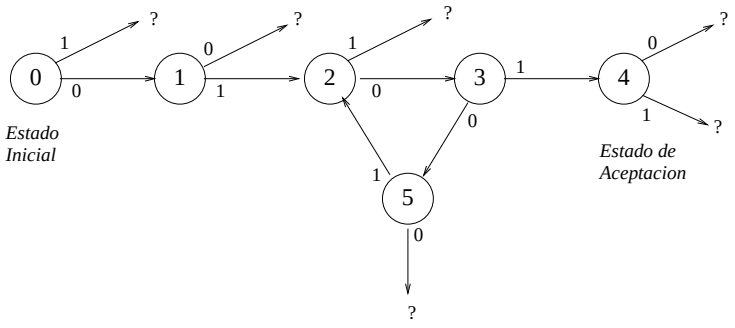


Figura 1.2: Un posible diagrama de transición de estados

Las transiciones representan a la función δ. Por ejemplo, si el autómata se encuentra en el estado 3, entonces δ determina que el siguiente estado es 4 si hay una entrada 1, o es 5 si la entrada es 0.

Sin embargo, ¿cómo *se sabe* que este diagrama es correcto? Desafortunadamente, no se sabe. Es fácil imaginar otros diagramas que den exactamente el mismo resultado, incluyendo aquél que se presenta en la figura 1.3.

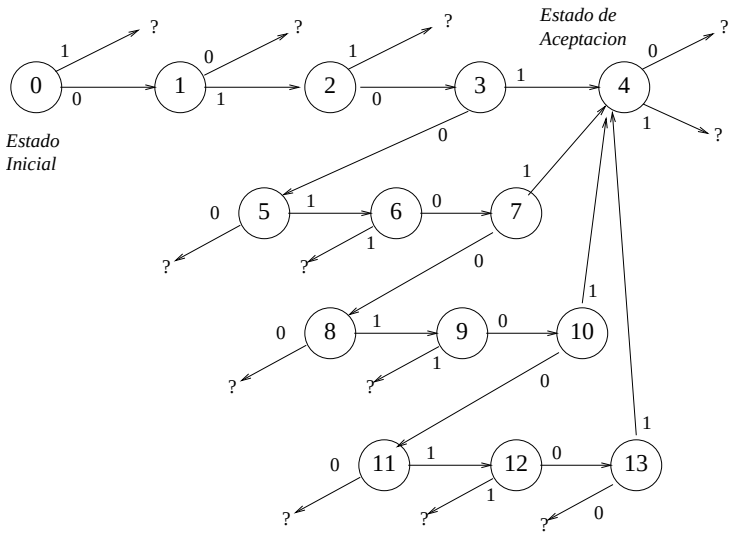


Figura 1.3: Otro posible diagrama de transición de estados

También es posible estar errado respecto a la suposición de que la caja negra acepta todas las cadenas de la forma $01(001) * 01$. Podría aceptar tales cadenas, por ejemplo, hasta 98 repeticiones de 001, pero no con 99 repeticiones. Analizar un dispositivo desconocido como caja negra puede resultar mucho muy complicado.

En este punto, alguien del equipo de investigación podría haber volteado la caja negra y descubrir una etiqueta con la siguiente información:

```
Caja Negra 1237-SWD
   (6 palabras)
```

Dada tal información, es finalmente posible determinar precisamente el lenguaje que acepta el autómata dentro de la caja negra. La teoría (véase el capítulo 3) dice que si un autómata acepta seis palabras de la forma $01(001)^n01$ (lo que significa 01 seguido por n repeticiones de 001 seguido por 01), para $n = 0, 1, 2, 3, 4, 5$, entonces aceptará todas las palabras del conjunto $01(001) * 01$. Se puede escribir entonces que:

$$01(001) * 01 \subseteq L_A$$

donde L_A representa al lenguaje del autómata A, que se encuentra dentro de la caja negra.

Una exploración sistemática de las posibilidades lleva a otra de tales conclusiones:

$$(0110) * 111 \subseteq L_A$$

Se puede escribir entonces que

$$01(001) * 01 + (0110) * 111 \subseteq L_A$$

lo que significa que una palabra ya sea de la primera forma o de la segunda forma será aceptada por el autómata A.

Ya que hay un número finito de posibilidades para tales secuencias, eventualmente se puede obtener una expresión completa del autómata en la caja; se puede ignorar cómo es exactamente su estructura interna, pero es posible saber precisamente cómo respondería a cualquier secuencia de entrada concebible.

El lenguaje que acepta un autómata finito puede siempre escribirse como una *expresión regular*, que se obtiene de aplicar las siguientes reglas:

- 0 y 1 son expresiones regulares.
- Si X y Y son expresiones regulares, también lo es su *concatenación* (XY), así como su *suma* $(X + Y)$.
- Si X es una expresión regular, también lo es su *iteración* $(X*)$.

Además, para toda expresión regular hay un autómata que acepta el lenguaje simbolizado por esa expresión. Por tanto, en cierto sentido, las expresiones regulares capturan precisamente el comportamiento de un autómata en términos de su lenguaje.

Sin embargo, para toda expresión regular hay un número infinito de autómatas que aceptan ese lenguaje. Así, aun cuando se conoce precisamente el lenguaje que un autómata acepta, es necesario abrir la caja e "inspeccionar" su diagrama de transición de estados (por decirlo de algún modo), si se desea saber precisamente que dispositivo se encuentra en ella.

Los autómatas finitos son la clase de modelos computacionales más sencillos, y han sido ampliamente estudiados. Como tales, tienen limitaciones que pueden no ser directamente aparentes. Por ejemplo, ningún autómata finito puede aceptar el conjunto de todos los "palíndromes marcados" sobre

un alfabeto dado. Estas son palabras de la forma zmz^{-1}, donde z es una palabra en algún alfabeto, z^{-1} es su reverso, y m es un símbolo especial de marca que no pertenece al alfabeto. En el capítulo 3 se da un ejemplo aun más simple de un lenguaje no regular. Por ahora, en el capítulo 2 se discuten dispositivos de cómputo más poderosos.

Capítulo 2

La Jerarquía de Chomsky
Cuatro Computadoras

A primera vista, un modelo computacional teórico podría no ser muy diferente al *hardware* de computadora. La diferencia básica puede ser que el primero es abstracto, mientras que el segundo es concreto. Sin embargo, un modelo computacional teórico permite delinear precisamente el poder y limitaciones de un esquema de computación específico. Aun cuando muchos modelos computacionales han sido propuestos desde el inicio de la computación, todos tienden a ser equivalentes a uno de los cuatro principales modelos: autómata finito, autómata de pila, autómata lineal y máquinas de Turing. En el presente capítulo, estos cuatro modelos computacionales son vistos como variaciones de una sola máquina, como la que se muestra en la figura 2.1

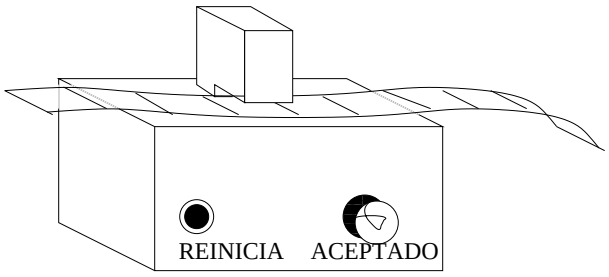


Figura 2.1: La máquina básica

La caja representa un conjunto de reglas que determinan cómo responde la máquina al contenido de una cinta que pasa por su lector, una celda a la

vez. Cada celda contiene un símbolo perteneciente a un alfabeto finito. La máquina opera en ciclos. Cada ciclo consiste en leer un símbolo, responder, y entonces mover la cinta. La respuesta de la máquina determina, en general, cuál de los cuatro tipos de modelo es.

La jerarquía de Chomsky recibe este nombre en honor al filósofo y lingüista estadounidense Noam Chomsky. Cada modelo de cómputo determina una clase de lenguaje. Ya que cada modelo es más general que su predecesor en la jerarquía, cada clase de lenguaje incluye aquélla que le precede:

Generalidad	**Modelo de Cómputo**	**Clase de Lenguaje**
-	Autómata finito	Regulares
	Autómata de pila	Libres de Contexto
	Autómata lineal	Sensibles al contexto
+	Máquinas de Turing	Recursivos enumerables

Un ejemplo específico de cualquiera de los cuatro modelos puede bien ser llamado *computadora*. Tiene un número finito de estados, y cada ciclo de operación involucra una transición de un estado a otro que se dispara por un símbolo específico del alfabeto de la computadora. Los estados de la computadora puede representarse por círculos etiquetados, y las transiciones como flechas entre los círculos (figura 2.2).

Figura 2.2: Transición de un autómata

La computadora tiene un estado especial llamado *estado inicial*, y uno o más *estados de aceptación* ó *estados finales*. Cuando una palabra x se coloca en la cinta al principio de la operación de la computadora, la computadora se encuentra en su estado inicial. Procede entonces a responder a la cinta, símbolo por símbolo. Si el foco de "ACEPTADO" se enciende tras que la computadora lee el último símbolo de x, entonce se encuentra en un estado de aceptación. Esto significa que la computadora ha aceptado x. Por tanto, el *lenguaje* de la computadora es el conjunto de todas las palabras que ésta acepta.

El modelo de la computadora genérica que se ha descrito se determina enteramente por la naturaleza de la respuesta a los símbolos en la cinta. Por

ejemplo, si la cinta se mueve en una sola dirección y la respuesta consiste meramente en el avance de la cinta seguido por una transición a otro estado, entonces la computadora es un *autómata finito*.

Sin importar el modelo que se trate, no hay nada que prohiba que la computadora escriba en la cinta. Cuando la cinta es unidireccional, la presencia o ausencia de tales símbolos de salida no hace ninguna diferencia respecto al lenguaje que la computadora acepta. Sin embargo, se hace distinción de esta variante de un autómata finito como una *máquina Mealy*, nombrada así en honor al matemático G.H. Mealy. En tal modelo, las transiciones en la computadora se representan como se muestra en la figura 2.3. Si la computadora está en el estado i y el símbolo que se lee es a, entonces la computadora escribe un símbolo b, avanza la cinta, y entra al estado j.

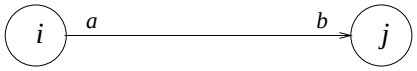


Figura 2.3: Transición en una máquina Mealy

El siguiente nivel de generalidad en la jerarquía de Chomsky involucra *autómatas de pila* (figura 2.4). Aquí, se altera temporalmente la máquina genérica para incluir una cinta adicional.

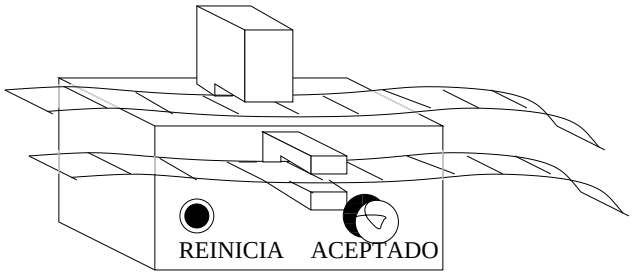


Figura 2.4: Autómata de pila

Una computadora de este tipo se restringe a una cinta principal unidireccional tal y como es un autómata finito. La cinta auxiliar, sin embargo, puede moverse hacia adelante o hacia atrás.

Inicialmente, este tipo de computadora revisa tanto la primera celda de la cinta auxiliar como el primer símbolo de la palabra en la cinta principal. En cualquier punto del cómputo, un ciclo consiste en leer los símbolos en

ambas cintas, y reponder de acuerdo a ellos. Antes de avanzar sobre la cinta principal y entrar en un nuevo estado, la computadora puede *(a)* avanzar sobre la cinta auxiliar y escribir un símbolo en la siguiente celda, o *(b)* borrar el símbolo existente en la celda actual de la cinta auxiliar, y regresar sobre la cinta auxiliar una celda.

La primera celda de la cinta auxiliar contiene un símbolo especial que la computadora reconoce como una marca. No se permite escribir sobre esta marca, o mover la cinta de regreso a partir de ella; hacerlo obviamnte requeriría leer una celda no existente sobre la cinta auxiliar en el siguiente ciclo.

En forma de diagrama, un autómata de pila puede tener dos tipos de transición a partir de un estado dado (figura 2.5). Los términos *push* y *pop* se refieren a una visualización común de la cinta auxiliar, conocida popularmente como "pila". Los símbolos sobre esta cinta se "apilan" en el siguiente sentido: para leer un símbolo s dado en esta cinta, la computadora debe leer y borrar todos los símbolos entre s y la celda actualmente revisada de la propia cinta. Esto significa remover todos los objetos de una pila hasta obtener un objeto específico de enmedio.

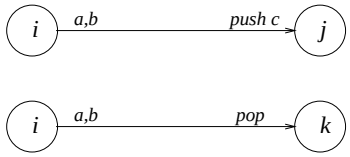


Figura 2.5: Dos tipos de transición en un autómata de pila

Se considera que un símbolo c que se añade a la cinta auxiliar es equivalente a añadirlo (*"pushed"*) a la pila. Si un símbolo se remueve de la pila (*"popped"*) no se requiere especificar de cuál se trata: es cualquier símbolo que se encuentre en la celda que actualmente se revisa.

Un autómata de pila tiene una característica adicional no compartida con los autómatas finitos. Dado un estado actual y un par de símbolos de las cintas principal y auxiliar, un autómata de pila no se encuentra confinado a un estado siguiente específico. Puede dirigirse a uno de varios estados (figura 2.6). Tales transiciones se conocen como *no-determinísticas*, ya que el siguiente estado no es determinado únicamente por las condiciones actuales. Cuando se trata de aceptar palabras de su lenguaje, un autómata de pila se

supone que siempre hace una transición correcta, esto es, una transición que siempre lleva a un estado de aceptación. Cuando procesa una palabra que no se encuentra en su lenguaje, sin embargo, no importa cómo la computadora añada o saque de su pila, o qué transición tome cuando se le ofrece una selección: nunca puede ser un estado de aceptación cuando el último símbolo de su cinta de entrada se procesa.

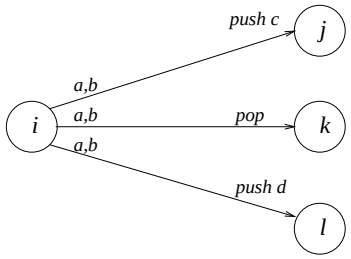


Figura 2.6: Una transición no-determinística

De acuerdo con esto, se definen dos tipos de autómatas de pila: determinísticos y no-determinísticos. Por definición, la segunda clase de autómatas incluye a la primera. El conjunto de lenguajes que acepta un autómata de pila (no-determinístico) es conocido como "libre de contexto". Los lenguajes que se aceptan por un autómata de pila determinístico se conocen como "determinísticos libres de contexto".

Obviamente, la clase de autómatas de pila incluye a la clase de autómatas finitos: cualquier autómata finito puede convertirse en un autómata de pila si se le equipa con una cinta auxiliar, que el autómata finito ignora para efectos prácticos. El autómata finito se mantiene simplemente escribiendo el mismo símbolo en su cinta auxiliar mientras que lee y responde a los símbolos en su cinta principal.

Un *autómata lineal*, como un autómata de pila, puede operar en forma no-determinística, pero a diferencia del autómata de pila, no está equipado con una cinta auxiliar. En lugar de esto, tiene una sola cinta principal como el autómata finito. Sin embargo, el autómata lineal sí tiene la capacidad de leer y escribir sobre su cinta. Como se muestra anteriormente en el caso de la máquina Mealy, esto no da ninguna ventaja especial al autómata lineal, a menos que pueda mover su cinta en ambas direcciones. La respuesta de un autómata lineal puede, por lo tanto, indicarse en forma de una transición

como se muestra en la figura 2.7. En esta figura, la computadora se encuentra en el estado i, en el cual revisa su cinta por el símbolo a. Escribe el símbolo b para remplazar la a, y mueve la cinta en la dirección D. En la transición que se muestra, D puede tomar valores de L para izquierda, R para derecha, y S para detenerse.

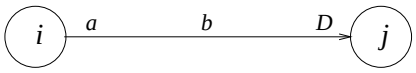


Figura 2.7: Una clase más general de transición

De hecho, si la definición de autómata lineal se complementa con una condición, la computadora que se define es ahora una *máquina de Turing*, el modelo computacional más general que se conoce. La condición es: para cualquier palabra de entrada x de longitud n (es decir, que contiene n símbolos), se le permite a un autómata lineal una constante k veces del total de su cinta para completar su cómputo. Por ejemplo, para un autómata lineal dado la constante k podría ser 5. Para decidir si una palabra de longitud 7 se acepta, se le permitiría al autómata lineal el uso de 35 celdas consecutivas de su cinta, y ni una más. Siete de estas celdas estarían ocupadas por la palabra, y las restantes 28 celdas podrían usarse como área de trabajo.

Un autómata lineal acepta la clase de lenguajes llamados "sensibles al contexto", mientras que las máquinas de Turing aceptan los lenguajes llamados "recursivamente enumerables".

La clase de autómatas lineales incluye la clase de autómatas de pila. Este hecho no es tan observable a simple vista como la discusión de que los autómatas de pila incluyen a los autómatas finitos. Dado un autómata de pila M, se podría sin embargo, construir en forma relativamente fácil un autómata lineal que lo simule. La clave para la simulación recae en el hecho de que un autómata de pila usa una cantidad de cinta auxiliar (la pila, propiamente) que se limita linealmente por el tamaño de su palabra de entrada. Conforme el autómata de pila procesa esta palabra, símbolo por símbolo, utiliza a lo más un número fijo k de celdas en la cinta auxiliar. El autómata lineal M' que simula a M utiliza su espacio de trabajo para imitar la cinta auxiliar. Se mueve de ida y vuelta entre la palabra de entrada y el espacio de trabajo, alternativamente representando operaciones en la cinta principal y auxiliar.

En la definición dada de una máquina de Turing queda implícitamente supuesto que son no-determinísticas. Como se definen normalmente, las máquinas de Turing son determinísticas. Para cada estado actual y combinación de símbolos de entrada, hay sólo una respuesta abierta para este modelo de cómputo. Ciertamente, esto no hace diferencia. Se podría, con algo de paciencia, construir una máquina de Turing determinística equivalente para otra no-determinística arbitraria. Desafortunadamente, nadie sabe si los autómatas lineales determinísticos son equivalentes a los no-determinísticos.

Este capítulo hace énfasis en la actuación de varios estilos de computadoras abstractas que "aceptan" lenguajes. De hecho, las computadoras se consideran más comúnmente como transductores de entrada/salida. La palabra en la cinta al principio del cómputo se transforma cuando (y solo si) la computadora se detiene en una nueva palabra. Tales palabras pueden representarse por números, programas de computadora, o virtualmente cualquier entidad simbólica bien definida.

Capítulo 3

Lenguajes Regulares
Bombeando Palabras

Además de los lenguajes de programación, existen otros lenguajes más abstractos que se estudian dentro de la computación. Sus nombres dan una leve idea sobre su naturaleza (véase el capítulo anterior):

- Lenguajes regulares
- Lenguajes libres de contexto
- Lenguajes sensibles al contexto
- Lenguajes recursivamente enumerables

En la clasificación de lenguajes abstractos, es frecuentemente útil saber cuando un lenguaje dado *no* pertenece a una clase particular. Una técnica para hacer precisamente esto se basa en el "lema de bombeo" (*pumping lemma*), publicado por primera vez en 1961 por Y. Bar-Hillel, M. Peries y E. Shamir. El lema del bombeo para lenguajes regulares dice que si se selecciona una palabra lo suficientemente larga de un lenguaje regular y se "bombea" (tantas veces como se desee), siempre se obtiene una nueva palabra en tal lenguaje.

Lo que hace tal conocimiento útil es que en ocasiones, al tratar con una palabra en un lenguaje del que se conoce poco, se puede hallar que una palabra particular en él no puede "bombearse" en ninguna forma sin obtener una palabra que *no* pertenece al lenguaje. En tal caso, se sabe inmediatamente que el lenguaje no es regular.

"Bombear" una palabra W significa seleccionar una sub-palabra particular Y de W, y reemplazar Y por YY en W. Simbólicamente, si

$$W = XYZ$$

entonces el resultado de "bombear" W (en Y) una vez es

$$W' = XYYZ$$

y el resultado de "bombear" W dos veces es

$$W'' = XYYYZ$$

y así en adelante. Ciertamente, se puede "bombear negativamente" una palabra W mediante eliminar Y de W.

Es posible descubrir lo que dice el lema de bombeo y cómo trabaja en forma simultánea, mediante examinar una porción (figura 3.1) del diagrama de transición de estados de un autómata finito A que acepta un lenguaje regular L en particular (véase el capítulo 2). Como es costumbre, los diferentes estados del autómata se representan por círculos, y las transiciones entre estados se representan por flechas etiquetadas con el símbolo de entrada que causa esa transición en particular.

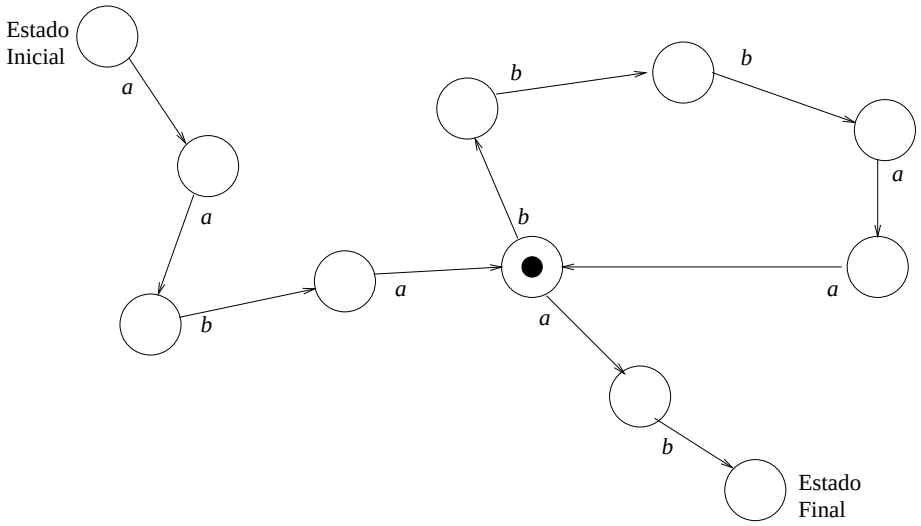


Figura 3.1: Una palabra de entrada lleva un camino

Cada palabra en L corresponde a un "camino" a través del diagrama de A. El camino lleva de un estado inicial a alguno de los estados finales de A. Por ejemplo, el camino para la palabra *aababbbaaab* se muestra en la figura 3.1. El camino es simplemente la secuencia de transiciones por la que debe ir A a fin de aceptar una palabra.

Lo más interesante acerca del camino de la figura anterior es que contiene un ciclo: habiendo entrado a uno de los estados de A, el camino regresa más tarde a él. Esta observación da inmediatamente una idea para producir nuevas palabras que A debe aceptar. Simplemente, repite la porción de la palabra que corresponde al ciclo:

$$aaba(bbbaa)ab$$

para obtener

$$aaba(bbbaa)(bbbaa)ab$$

En otras palabras, *aababbbaabbbaaab* debe también ser aceptada por A, y por lo tanto, debe estar en L. Mediante un razonamiento similar, es posible ir por el ciclo tantas veces como se desee antes de llegar a un estado final. Cada vez, se obtiene una nueva palabra de L.

Desafortunadamente, no hay nada que garantice que el camino correspondiente a una palabra arbitrariamente seleccionada de L tenga un ciclo. Sin embargo, A tiene sólo un número n finito de estados, y una palabra de longitud mayor o igual a n que resulte estar en L ciertamente lleva a A a través de un ciclo: cualquier palabra de longitud n o más debe causar que al menos un estado vuelva a visitarse. Esta simple observación es un ejemplo del "principio de la casilla" (*pigeonhole principle*): si más de n cartas se colocan en n casillas de correo, entonces al menos una casilla debe recibir más de una carta.

Una observación más permite llegar a la posición de enunciar el lema de bombeo. Nótese que la sub-palabra:

$$XY = aababbbaa$$

donde $X = aaba$ y $Y = bbbaa$, tiene una longitud total menor o igual a n, mientras que la sub-palabra Y tiene longitud mayor o igual a 1.

El Lema de Bombeo para Lenguajes Regulares

Si L es un lenguaje regular, entonces hay una constante n tal que por cada palabra W en L con longitud $\geq n$, hay palabras X, Y, Z tales que:

- $W = XYZ$
- Longitud de $XY \leq n$
- Longitud de $Y \geq 1$
- XY^kZ está en L para $k = 1, 2, 3, ...$

Aquí, Y^k simplemente significa k repeticiones de la sub-palabra Y concatenadas en la forma como se menciona anteriormente.

Resulta interesante utilizar el lema de bombeo para mostrar que un lenguaje particular es *no* regular. Por ejemplo, sea L un lenguaje consistente de todos los palíndromes sobre el alfabeto $[a, b]$. Tales palabras son simétricas respecto a su punto medio, por ejemplo:

$$abbababba$$

Si L es regular, entonces el lema de bombeo se aplica a L y debe haber una constante n (en general, desconocida) la cual debe usarse para medir la longitud de las palabras candidatas para aplicárseles el lema de bombeo. La estrategia a usar aqui es notar que no importa qué valor pueda tener n, la palabra:

$$W = a^n b a^n$$

debe estar en L de acuerdo con la definición de los palíndromes. Pero de acuerdo con el lema de bombeo, W puede ser escrita como:

$$W = XYZ$$

de tal modo que, en particular, la longitud de XY como la parte inicial de W es menor o igual a n. Por lo tanto, XY consiste nada más de a's, y Y consiste de al menos una a. Sigue entonces, por el lema de bombeo, que la palabra:

$$W = XY^2Z = a^m b a^n$$

también está en L. Ahora, a^m refleja el hecho de que alguna porción no nula de la cadena inicial a^n ha sido bombeada, así que $m < n$. Pero en este caso $a^m b a^n$ no puede estar en L, ya que no es un palíndrome. Lo único que puede haber fallado en todo el razonamiento anterior es la suposición inicial de que L es un lenguaje regular. Evidentemente, no lo es.

Hay lemas de bombeo (no exactamente iguales al lema de los lenguajes regulares, pero similar en espíritu) para otros tipos de lenguajes abstractos, incluyendo los lenguajes libres de contexto.

Capítulo 4

Gramáticas Generativas
Sistemas Lindenmeyer

El crecimiento de ciertos tipos de plantas puede ser modelado, hasta cierto punto, por un esquema formal conocido como los *sistemas Lindenmeyer*. Tales sistemas, desarrollados inicialmente por el biólogo y matemático Aristid Lindenmeyer en 1968, son realmente una clase especial de gramática generativa. En este contexto general, las palabras de un lenguaje formal se producen por un proceso de reemplazo paso a paso. Comenzando por un solo símbolo, a cada paso uno o más símbolos de la palabra actual se reemplazan por ciertas palabras dadas en un lista de "producciones". Cuando la palabra ha cesado de "crecer", se le considera como un miembro del lenguaje generado por la gramática.

Una ilustración simple y gráfica de este proceso se ejemplifica por el crecimiento de la alga roja (figura 4.1), provisto por Lindenmeyer.

En este ejemplo, muchos de los pasos intermedios se han omitido. De hecho, comenzando por el "brote" inicial que consiste en una sola célula, el modelo de alga tiene que pasar por seis divisiones antes de que alcance el primer estado de la figura. Tales divisiones se muestran enh la figura 4.2.

Más adelante se retoman los sistemas Lindenmeyer y este ejemplo en particular, pero antes es necesario discutir algunas propiedades notables de las gramáticas generativas.

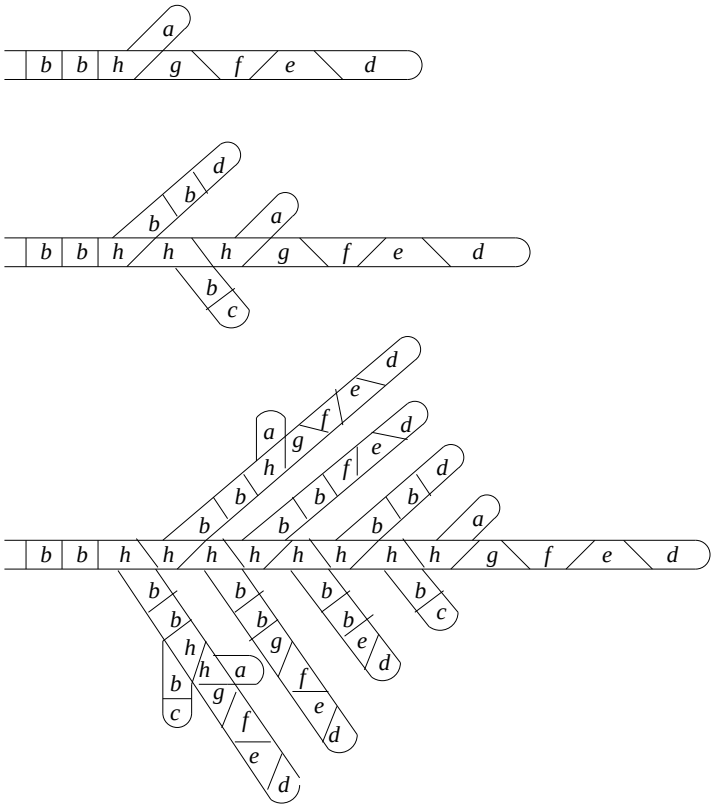


Figura 4.1: Crecimiento de la alga roja

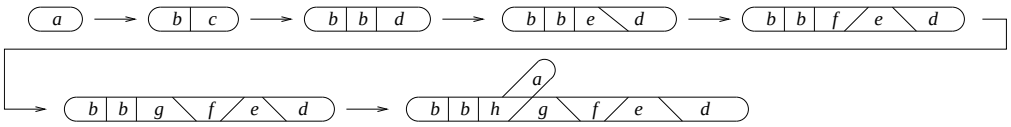


Figura 4.2: Los primeros seis pasos de crecimiento

Una gramática generativa G consiste de una cuádrupla (N, T, n, P), donde:

- N es el alfabeto de los *símbolos no-terminales*
- T es el alfabeto de los *símbolos terminales*
- n es el *símbolo inicial*, $n \in N$
- P es el conjunto de *producciones*

Los alfabetos N y T son disjuntos, y normalmente se refiere a su unión como el conjunto A. La gramática G genera palabras por medio de las producciones en P. Cada producción es un par ordenado (X, Y), donde X y Y son palabras sobre el alfabeto A; formalmente, se escribe $X, Y \in A^*$. Más aun, la palabra X debe contener al menos un símbolo no-terminal. La producción (X, Y) se escribe normalmente en la forma $X \to Y$, con la intención de reemplazar la ocurrencia de la palabra X (en una palabra más larga) por Y. Una palabra W "genera" otra palabra W':

$$W \Rightarrow W'$$

si W tiene la forma $W_1 X W_2$, W' tiene la forma $W_1 X' W_2$, y $X \to X'$ es una producción en P. Una secuencia de tales generaciones, que se denota $W^* \Rightarrow W'$ se llama *secuencia de derivación*, siendo la última palabra "derivada" de la primera. El conjunto de todas las palabras obtenibles de esta forma a partir de un símbolo inicial se le llama *lenguaje generado por G*. En forma de notación:

$$L(G) = \{W : W \in T^*, n^* \Rightarrow W\}$$

Por ejemplo, el conjunto de todos los palíndromes sobre el alfabeto $\{0, 1\}$ se produciría por la siguiente gramática:

$$N = \{n\}, T = \{0, 1\}, n, P = \{n \to 0n0, n \to 1n1, n \to 0, n \to 1, n \to \lambda\}$$

La ilustración más simple de este hecho es un árbol de derivación que muestra todas las posibles palabras de una longitud dada que resultan de la gramática (figura 4.3). Por lo tanto, hay dos palíndromes de longitud 1, dos de longitud 2, cuatro de longitud 3, etc. Nótese cómo cada nueva palabra se obtiene mediante reemplazar el símbolo no-terminal n por una de las cinco palabras en el conjunto de producción. Recuérdese que en general se

reemplaza una sub-palabra de una palabra dada, y que este ejemplo es un caso muy especial de esa regla.

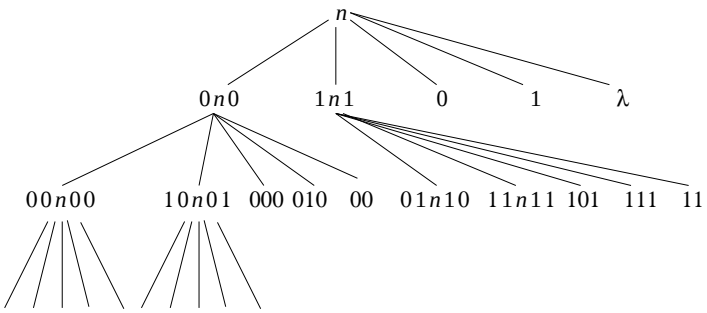


Figura 4.3: Arbol de derivación de palíndromes

En el capítulo 2 se menciona que existen cuatro tipos de autómatas: el autómata finito, el autómata de pila, el autómata lineal, y la máquina de Turing. También se menciona que hay cuatro tipos de lenguajes asociados con cada uno de estos autómatas, y que algunos de estos lenguajes tienen descripciones simples en términos teóricos puramente lingüísticos. En este punto, se puede considerar que también hay definiciones simples para cuatro clases diferentes de gramáticas generativas, cada una más general que la anterior, y que generan precisamente los lenguajes que se aceptan respectivamente por los autómatas descritos previamente.

Sea G una gramática en la cual cada producción tiene la forma $x \to yX$ ó $x \to X$, donde $x, y \in N$ y $X \in T^*$. Tal gramática resulta generar un lenguaje regular, lo que se prueba a continuación. Mientras tanto, nótese que las producciones usadas para crear palíndromes son de un tipo más general que aquéllas empleadas para la clase de gramática definida anteriormente. Al mismo tiempo, ningún autómata finito puede aceptar palíndromes (véase el caítulo 3).

Sea G una gramática en la que todas sus producciones presentan la forma $x \to X$, donde $x \in N$ y $X \in A^*$. Ya que X es una palabra sobre el alfabeto A (la unión de los conjuntos de símbolos terminales y no-terminales), esta clase de producción incluye la producción anteriormente descrita como un caso especial. También incluye una gramática generadora de palíndromes como caso especial.

Si ahora se expande el conjunto de los tipos de producciones permitidas aún más para incluir aquéllas de la forma $XyZ \rightarrow XYZ$, donde $X, Y, Z \in A^*$ y $y \in N$, se obtiene un tipo todavía más general de gramática G. El único requerimiento adicional de esta gramática es que la palabra Y, que reemplaza a y, no sea vacía a menos que la producción tenga la forma $y \rightarrow \lambda$, y que y no sea la palabra del lado derecho en ninguna producción de G.

Nótese que en esta última definición la producción esencial es $y \rightarrow Y$, pero que este reemplazo tiene lugar dentro de un cierto contexto , llamado $X...Y$. Es por esta razón que G es llamada *gramática sensible al contexto*. Naturalmente, el segundo tipo de gramática definido en el párrafo anterior, al no tener un contexto, se conoce como *gramática libre de contexto*.

Los lenguajes generados por esta gramáticas son los que se han denominado anteriormente como libres de contexto y sensibles al contexto (véase el capítulo 2). Tales lenguajes son aceptados por autómatas de pila y autómatas lineales, respectivamente.

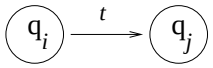
La cuarta y última gramática a ser considerada ya ha sido definida. Es la gramática generativa en su forma más general con ninguna restricción sobre el tipo de producción permitida (excepto aquéllas impuestas por la propia definición). Ya que no hay restricciones, tal gramática puede ser (y de hecho, es) llamada *gramática tipo 0*. De la misma forma, las gramáticas sensibles al contexto, libres de contexto y regulares también se les conoce como *tipo 1*, *tipo 2* y *tipo 3*, respectivamente. De nuevo, las gramáticas tipo 0 también corresponde a un autómata específico, pero solo del tipo más general; los lenguajes generados por gramáticas tipo 0 son aquéllos aceptados por máquinas de Turing.

En seguida se presenta una demostración de la equivalencia gramática/lenguaje en el caso más simple, en el cual gramáticas tipo 3 generan lenguajes regulares, que se aceptan por autómatas finitos.

De acuerdo con un problema expuesto en el capítulo 3, si un lenguaje L es regular, también lo es su *reverso*, es decir, el lenguaje que se obtiene de invertir todas las palabras de L. Este hecho es conveniente durante la demostración de que para todo lenguaje de tipo 3 (un lenguaje generado por una gramática de tipo 3), hay un autómata finito determinístico que lo acepta.

Primero, sea M un autómata finito determinístico que acepta entradas binarias. Para mostrar que el lenguaje L aceptado por M es tipo 3, se

construye una gramática G_M como sigue: Sea $N = \{q_0, q_1, ..., q_m\}$ el conjunto de estados de M, donde q_0 es el estado inicial. Sea el conjunto $T = \{0, 1\}$, y por cada transición



se escribe $q_i \rightarrow tq_j$, donde t es el símbolo de entrada, y q_i, q_j son estados de M. Se define a P como el conjunto de todas las producciones que se obtienen de esta manera. Si se añade la transición $q_k \rightarrow \lambda$ por cada estado final q_k, y se hace q_0 el símbolo inicial no-terminal, entonces la gramática G_M genera el reverso de L.

Por ejemplo, si la palabra 110 se acepta por M, entonces se puede ilustrar una serie de transiciones como las que se muestran en la figura 4.4. Las cuatro producciones en la figura 4.4 dan lugar a la secuencia de derivación $q_0 \Rightarrow q_1 1 \Rightarrow q_2 11 \Rightarrow q_3 011 \Rightarrow 011$. Esto es el reverso de 110.

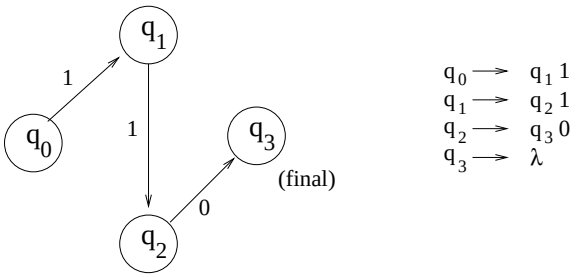


Figura 4.4: Convirtiendo transiciones en producciones

En general, si la palabra que acepta M se escribe como $a_1 a_2 ... a_n$ y si los estados de M a partir de q_0 y hasta el estado finales se denota como $q_1, q_2, ...q_n$ (este último, el estado final), entonces claramente se cumple que $q_0 \Rightarrow^* a_n ... a_2 a_1$ por extensión de la discusión anterior.

Si, sin embargo, se comienza con un lenguaje L tipo 3, entonces es relativamente fácil construir un autómata M_L que acepte el reverso de L. Para probar esto, es conveniente permitir que M_L sea un autómata no-determinístico, que es equivalente para los propósitos actuales a un autómata determinístico.

Dada una gramática G que genera a L, para cada producción $x \rightarrow yX$ ó $x \rightarrow X$ se establecen transiciones en un autómata correspondiente M_L cuyos estados incluyen los símbolos no-terminales de G. Estados adicionales se incluyen de acuerdo a lo siguiente: para acomodar una producción de la forma $x \rightarrow yX$ no sólo se utiliza x y y como estados q_x y q_y respectivamente, sino que también se añaden estados que permiten "procesar" (por así decirlo) a la palabra X. Supóngase que $X = x_1x_2...x_n$. Entonces, se añaden $n-1$ estados intermedios entre q_x y q_y en la forma expuesta en la figura 4.5. Las transiciones que no son símbolos de los que X se compone se dirigen a un mismo estado "terminal". Similarmente, se codifica una producción de la forma $x \rightarrow X$ mediante una cadena de estados, el último de los cuales se define como un estado final para M_L. La razón por la cual se requiere que M_L sea no-determinístico es que dado un estado como q_x en la figura puede bien tener varias transiciones a partir de él que se disparan por el símbolo de entrada x_n. En cualquier caso, la construcción muestra claramente porqué M_L acepta palabras de L, y sólo esas palabras.

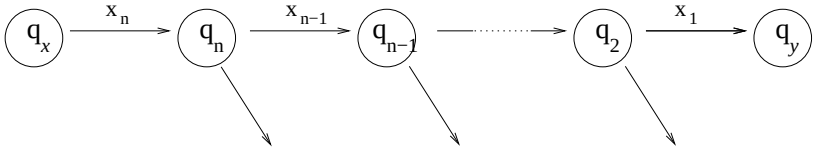


Figura 4.5: Convirtiendo producciones a transiciones

Probar que lenguajes de tipo 2, 1 y 0 son aquéllos aceptados por autómatas de pila, autómatas lineales y máquinas de Turing, respectivamente, es tan solo un poco más difícil que la prueba anterior.

Las gramáticas generativas se usan en la teoría y práctica de compiladores. Las palabras de un lenguaje de programación, conjuntamente con los símbolos para definir variables, arreglos y otras entidades del programa, son en sí símbolos en el lenguaje correspondiente. Esta descripción de lenguaje de programación no es en la que se conoce normalmente. En este caso, cada "palabra" en el lenguaje es en sí misma un programa sintácticamente correcto.

Regresando a los sistemas Lindenmeyer con los que se inicia este capítulo, es notorio que las producciones usadas para modelar la alga roja son:

$$\begin{array}{lll} a \to b|c & e \to f & (\to (\\ b \to b & f \to g &) \to) \\ c \to b|d & g \to h(a) & | \to | \\ d \to e/d & h \to h & / \to / \end{array}$$

Las células y las paredes entre ellas se indican simbólicamente por letras minúsculas, líneas verticales, líneas diagonales, y paréntesis. La línea vertical representa una pared recta, las diagonales una pared inclinada, y los paréntesis un nuevo brote. Este ejemplo tiene sólo un defecto: la persona o dispositivo que crea el diagrama del estado resultante de crecimiento debe recordar alternar la dirección de las paredes inclinadas y considerar brotes en los lados más largos de las células h.

Capítulo 5

Autómatas Celulares
El Juego de la Vida

En la obscuridad de un cuarto de proyección, se observa en la pantalla un patrón evolutivo gradual de pequeños cuadros. Dentro de este patrón, algunas poblaciones de cuadros parecen crecer mientras que otras parecen dirigirse a la extinción. A esto se debe el término de *vida* dado por su autor, John Conway, un matemático de la Universidad de Cambridge, a este juego que ha intrigado a miles de personas. Tras ser descrito por primera vez en la revista *Scientific American* de octubre de 1970 por Martin Gardner, el juego rápidamente se estableció como un pasatiempo importante de estudiantes de computación que tenían acceso a una computadora gráfica. Hasta algunos miembros académicos fueron atraidos a los círculos mágicos que rodeaban estas brillantes escenas, viendo tal vez un patrón similar al que se muestra en la figura 5.1.

El Juego de la Vida es un ejemplo de un autómata celular. Formalmente, debe pensarse en una malla cuadriculada infinita en la cual las células existen en uno de dos estados: "viva" o "muerta". Cada célula es un autómata simple que por cada tiempo de un gran reloj debe decidir en qué estado permanecer hasta el siguiente periodo de tiempo. Hace esta decisión en base no sólo del estado presente, sino también del estado de sus ocho células vecinas: cuatro adyacentes sobre sus lados y cuatro adyacentes en sus esquinas. En seguida, se presentan las reglas en que se basa la decisión:

- Si la célula se encuentra viva en el tiempo t, permanecerá viva durante el tiempo $t + 1$ si no está "sobrepoblada" o "desnutrida". En otras palabras, debe tener al menos dos vecinas vivas, y no más de tres.

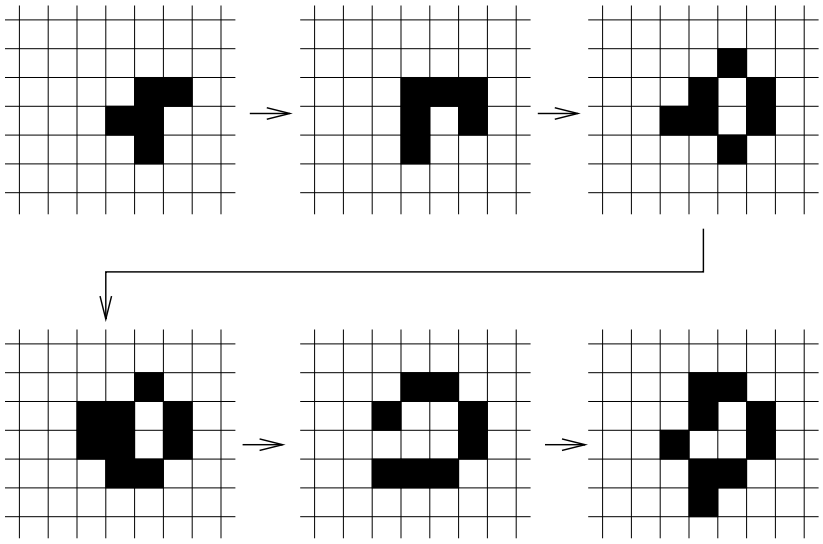


Figura 5.1: Seis generaciones de vida

- Si la célula está muerta en el tiempo t, permanecerá muerta durante $t + 1$ a menos que tenga tres "progenitores". Esto es, la célula debe tener tres células vecinas vivas para volver a nacer.

Aun cuando esta reglas parecen arbitrarias, en relidad no fue fácil para Conway descrubrirlas y proponerlas de tal modo que el comportamiento de la población de células vivas es tan difícil de predecir como fuera posible. Durante su desarrollo, Conway experimentó con docenas de reglas diferentes, descartando varios conjuntos de reglas tras experimentar con ellas. De entre las notas que envió a Gardner para el artículo de 1970, se incluía la predicción de que ninguna población podría crecer sin límite: tarde o temprano toda población debería extinguirse (es decir, todas las células permanecer eternamente muertas) o caer en un ciclo repetitivo y sin fin de patrones.

El Juego de la Vida puede jugarse manulamente en una malla cuadriculada usando fichas. Es difícil, sin embargo, jugar con un solo color de fichas, ya que al ir de una generación a la otra se debe recordar cuáles fichas estaban presentes en la generación anterior. Por esta razón, resulta mejor utilizar dos colores, uno para las células ya vivas (blanco, por ejemplo), y el otro para las células que acaban de nacer (gris, por ejemplo). Dada una generación t de fichas blancas, primero se colocan fichas grises en donde quiera que una

nueva célula nace. Después, se remueven todas las fichas blancas que representan a las células que mueren. Finalmente, se reemplazan las fichas grises por blancas.

Ciertamente, si se juega en una computadora, se pueden hacer muchas más cosas. Programar el Juego de la Vida es razonablemente sencillo. En ocasiones, conviene imprimir las generaciones sucesivas, especialmente para examinar generaciones pasadas con propósitos de análisis detallado. Fue usando una computadora que Conway y muchos otros pudieron encontrar nuevos e interesantes patrones. Un patrón inicial de células vivas puede introducirse a la computadora y aparecer en pantalla, y presionando una tecla, hacer que aparezcan generaciones sucesivas del patrón.

La figura 5.2 muestra *(a)* la evolución de una línea de siete células vivas de una "colmena" consistente de cuatro "abejas" (donde 13 generaciones intermedias no se muestran), y *(b)* una configuración de cinco células llamada "deslizador" (*"glider"*). El deslizador se repite a sí mismo cada cuatro ciclos, pero en una nueva posición.

Con tanta gente jugando el Juego de la Vida, no tomó mucho tiempo en que alguien descubriera un contraejemplo de la conjetura de Conway de que ninguna población pudiera crecer sin límite. Un grupo de seis estudiantes del Massachusetts Institute of Technology (MIT) descubrieron una configuración, la cual nombraron "pistola deslizadora" (*"glider gun"*), que emite un deslizador cada 30 generaciones. Un patrón que crece dentro de la pistola deslizadora después de 39 movimientos se muestra en la figura 5.3. Con deslizadores saliendo de la pistola cada 30 movimientos, el número total de células vivas obviamente crece sin límite. El mismo grupo de estudiantes logró arreglar 13 deslizadores que "chocaban" entre sí para formar una pistola deslizadora.

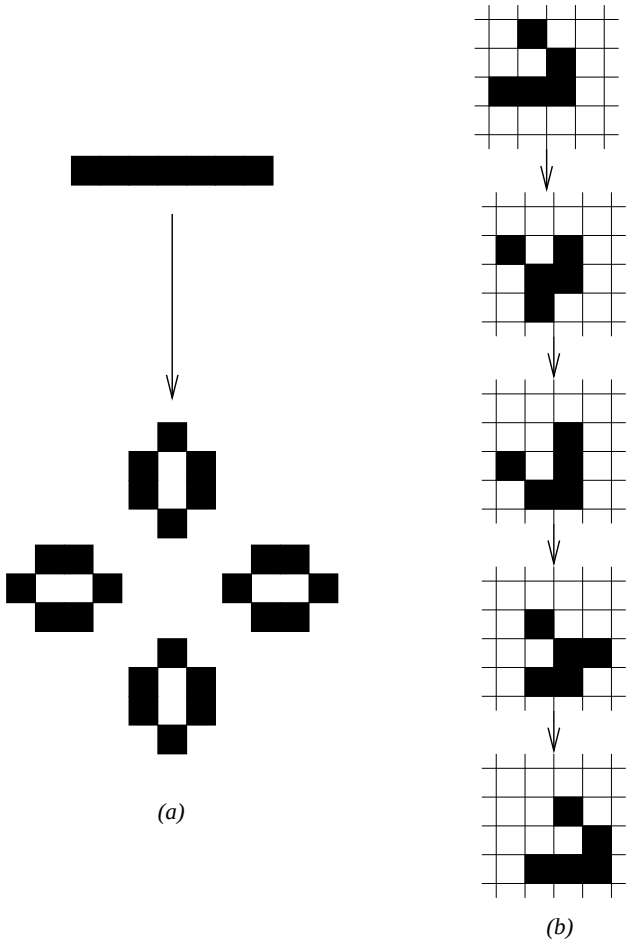


Figura 5.2: Evolución de *(a)* una "colmena" y *(b)* movimiento de un deslizador

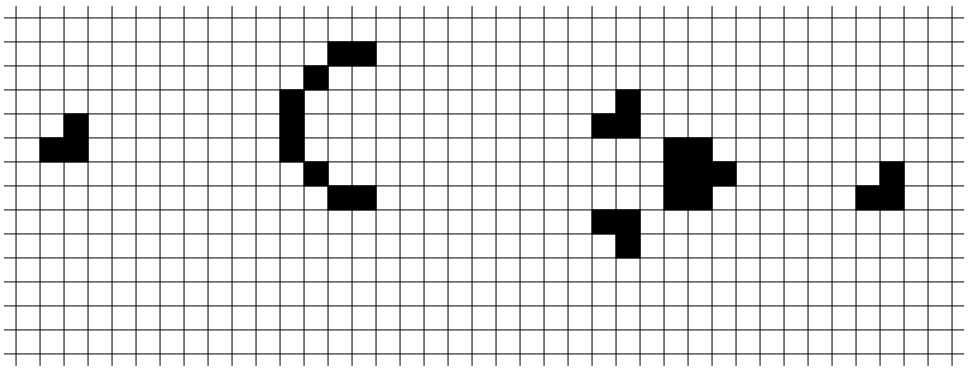


Figura 5.3: Una pistola deslizadora

El siguiente algoritmo calcula generaciones sucesivas del Juego de la Vida en una matriz L. Un "1" en la i,j-ésima entrada representa una célula viva, mientras que un "0" representa una célula muerta.

procedure *Life*

1. **for** $i \leftarrow 1$ **to** 100
   for $j \leftarrow 1$ **to** 100

   a) $s \leftarrow 0$

   b) **for** $p \leftarrow i-1$ **to** $i+1$ (calcula el efecto de los vecinos)
   for $q \leftarrow j-1$ **to** $j+1$
   $s \leftarrow s + L(p,q)$

   c) $s \leftarrow s - L(i,j)$

   d) **if** $(L(i,j) = 0 \bigwedge s = 3) \bigvee (L(i,j) = 1 \bigwedge (s = 3 \bigvee s = 4))$
   then $X(i,j) = 1$ (almacena vida o muerte en un arreglo X)
   else $X(i,j) = 0$

2. **for** $i \leftarrow 1$ **to** 100
   for $j \leftarrow 1$ **to** 100

   a) $L(i,j) \leftarrow X(i,j)$ (refresca L)

   b) **display** $L(i,j)$ (despliega L)

Al escribir un programa basado en este algoritmo, la sintaxis particular del lenguaje utilizado debe, por supuesto, substituir las flechas (que simbolizan asignaciones) e indentaciones (que indican enunciados **for** e **if**, por ejemplo).

Este algoritmo es relativamente sencillo. Los primeros dos ciclos **for** calculan nuevos valores para la *i,j*-ésima posición de L, mediante revisar el vecindario de (i, j). La variable s contiene la suma de estos valores (restando el valor de $L(i, j)$). El enunciado 1.d representa el criterio de vida o muerte mencionado anteriormente. El arreglo auxiliar X se utiliza para almacenar el nuevo valor de $L(i, j)$. Este valor no puede ser todavía reemplazado en L, ya que el valor original de $L(i, j)$ debe participar en otros cuatro cálculos de valores en L. Sólo hasta el paso 2.a los nuevos valores de $L(i, j)$ (que se encuentran en X) se actualizan. Al mismo tiempo, por así decirlo, los nuevos valores se despliegan en pantalla, mediante la línea 2.b.

Este algoritmo calcula tan solo una generación del Juego de la Vida. Debe ser introducido en otro ciclo exterior, que el programador puede crear y controlar a voluntad para calcular nuevas generaciones o terminar el programa.

Las configuraciones de células vivas que alcanzan el borde de la matriz de 100×100 mueren automáticamente. Esto podría prevenirse mediante el uso de "matrices circulares", que consideran aritmética de módulo para no salir de un conjunto de valores, haciendo por ejemplo, que el valor $L(100, 1)$ se encuentre adyacente a $L(1, 1)$. En tal caso, sin embargo, el cálculo de s en el ciclo 1.b debe modificarse para reflejar estas nuevas reglas de adyacencia entre células.

Cualquier programa basado en este algoritmo debe incluir las declaraciones e inicializaciones pertinentes y apropiados.

Los autómatas celulares han sido un área académica que ha fascinado a investigadores en computación y matemáticos desde el mismo John von Neumann. El objetivo de von Neumann al construir la primera computadora digital era construir una "máquina" auto-reproductiva. Al ser persuadido de que los espacios celulares eran un escenario ideal para sus experimentos, von Neumann logró desarrollar autómatas celulares en los que las células tenían cada una 29 estados. Además, conjeturó la existencia de una configuración de alrededor de 200,000 células que sería auto-reproductiva. Esto significa que cuando el autómata celular se colocaba en tal configuración, después de un periodo definido, daba como resultado dos configuraciones iguales a la original, una junto a la otra. A partir de los trabajos de von Neumann en autómatas celulares se han encontrado otros autómatas celulares auto-reproductivos, pero mucho más sencillos.

En términos formales, un autómata celular consiste de tres cosas:

1. Un *autómata* del cual una copia se asocia a cada célula en una malla infinita y n-dimensional.

2. Una *función de vecindario* que especifica cuáles de las células adyacentes a una célula determinada pueden afectarla.

3. Una *función de transición* que especifica para cada combinación de estados en las células vecinas cuál es el siguiente estado de la célula dada.

Además de construir autómatas auto-reproductivos, los investigadores en el área han desarrollado autómatas celulares que pueden realizar cálculos. Esto se hace comúnmente mediante incluir el equivalente a una máquina de Turing en un espacio celular: los patrones de movimiento de estados representan las acciones de la cabeza de lectura/escritura, y revisar los patrones de estados representa revisar la cinta. Existe, al menos en forma teórica, autómatas celular universales que pueden realizar cálculos, y algunos incluso son auto-reproductivos. De hecho, se puede comprobar que el Juego de la Vida mismo tiene esta propiedad.

Bibliografía

[1] E.R. Berlekamp, J.H. Conway, and R.K. Guy. *Winning Ways.* vol. 2, Academic, 1982.

[2] N.Chomsky. *Three models for the description of languages* IRE Trans. Information Theory 2, 1959.

[3] E.F. Codd. *Cellular Automata.* Academic, 1968.

[4] D.I.A. Cohen. *Introduction to Computer Theory.* Wiley, 1986.

[5] G.T. Herman and G. Rozenberg. *Developmental Systems and Languages.* North Holland, 1975 .

[6] J.E. Hopcroft and J.D. ULLman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[7] M. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.

[8] A.M. Turing. *On computable numbers with application to the Entscheidungsproblem.* Proc. London Math. Soc. 42, 1936. Erratum: Ibid. 43, 1937.

[9] A. Salomaa. *Formal Languages.* Academic, 1973.

[10] D. Wood. *Theory of Computation.* Harper and Row, 1987.

[11] S. Wolfram (editor). *Theory and Applications of Celluar Automata.* World Scientific, 1986.

[12] S. Wolfram. *Cellular Automata and Complexity: Collected papers.* 1994.