De lo Viejo a lo Nuevo[1] en Teoría de la Computación: Teoría de Lenguajes Formales y de Autómatas vs. Computación Molecular[2]

por

Carlos Martín–Vide y Gheorghe Păun[3]

El artículo presenta diversos ejemplos de nociones y resultados "antiguos"de la teoría de lenguajes formales y de la teoría de autómatas que han encontrado aplicaciones (a veces sorprendentes) en la muy reciente área de la computación molecular. Entre ellos, mencionaremos: el lenguaje de intercalación gemela (y la caracterización correspondiente de los lenguajes recursivamente enumerables), que tiene una relación directa con la estructura de las moléculas de ADN; los autómatas finitos de dos cabezas, que viven una nueva vida bajo el nombre de autómatas de Watson-Crick; la (prueba de la) llamada forma normal de Geffert, que ha conducido a un elegante modelo de la computación con ADN (que se conoce como modelo YAC); la forma normal binaria en el terreno de las gramáticas matriciales, que es una herramienta fundamental en la computación con membranas; etc. En muchos de los casos, han aparecido nuevos problemas sobre aquellos conceptos clásicos, lo cual ha revitalizado el interés por el estudio de éstos. El artículo requiere sólo un bagaje matemático general. Se han definido breve pero rigurosamente todos los conceptos que se emplean, tanto de la teoría de lenguajes formales y de autómatas como de la computación con ADN y con membranas.

Introducción

La teoría de lenguajes formales nació a mediados del siglo XX, con el propósito principal de modelizar la sintaxis de las lenguas naturales (Chomsky), pero su historia se puede hacer remontar hasta inicios del siglo (Thue). Su desarrollo ha estado estrechamente vinculado a las ciencias de la computación,

[1] *Y al Revés*

[2] Palabras clave: Computación con ADN, Teoría de lenguajes formales, Sistemas adherentes, Autómatas de Watson-Crick, Sistemas H, Computación con membranas, Sistemas P.

[3] Este trabajo ha sido posible gracias a una ayuda del Comité Científico de la OTAN en España, 2000-2001.

especialmente desde que se observó (Ginsburg y Rice) que las gramáticas independientes del contexto de Chomsky se corresponden con la forma de Backus-Naur utilizada para describir la sintaxis del lenguaje Algol 60. Algunas áreas importantes de la teoría son la reescritura regulada, la complejidad descriptiva, los sistemas de Lindenmayer, las gramáticas contextuales de Marcus, los sistemas de gramáticas ..., por mencionar solamente unas pocas. Su desarrollo, si bien siempre motivado por cuestiones procedentes de las ciencias de la computación (incluida la inteligencia artificial), de la lingüística, de la biología, etc., se produjo en gran medida como un fin *per se*, tal como acostumbra a suceder en otras ramas de las matemáticas. El lector puede ilustrarse sobre el particular en [40] y en las referencias bibliográficas allí contenidas.

Desde hace unos pocos años, una activa área de investigación en las ciencias de la computación es la dedicada a la computación molecular, es decir, a la utilización de materiales biológicos como sostén de las computaciones. El tema más conocido y publicitado es el del empleo de moléculas de ADN como *bio-chips*, una posibilidad sobre la que ya se especulaba en los años 50 y que confirmó experimentalmente L. Adleman en 1994. La esperanza principal es la de obtener mecanismos de computación con un extraordinario paralelismo que sean capaces de resolver problemas intratables en un tiempo razonable. Tras el experimento de Adleman vinieron varios otros (todos ellos, hasta hoy, abordan problemas "de juguete"), y en estos momentos hay una gran cantidad de investigadores (biólogos, informáticos, matemáticos) que están buscando alguna aplicación de la computación con ADN en la vida real. Ello ha originado una notable acumulación de investigaciones teóricas: se han propuesto muchos modelos de computación que se inspiran en el modo como la naturaleza "computa" al nivel genético y se ha formulado un número apreciable de resultados. Citamos aquí [34] para los detalles y la información bibliográfica.

Otra rama reciente y en rápido desarrollo de la computación molecular (teórica) es la llamada computación con membranas, la cual propone modelos de computación distribuida en paralelo (que reciben el nombre de sistemas P) inspirados en la manera como las células vivas procesan compuestos químicos, energía e información. Esta área de investigación se inició con [33], y en este momento están siendo investigadas numerosas variedades de sistemas P en busca de modelos tan realistas, potentes y eficientes (en términos del tiempo que necesitan para resolver problemas determinados) como sea posible. La bibliografía actual de este campo se puede encontrar en la página `http://bioinformatics.bio.disco.unimib.it/psystems`

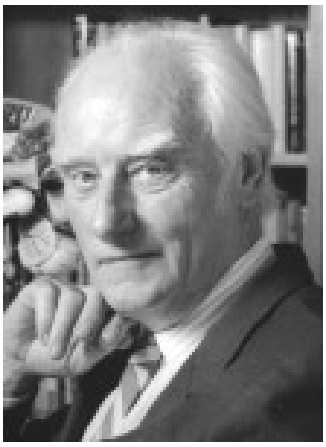
Como estructuras básicas de datos, una parte de los sistemas P emplean multiconjuntos de objetos-símbolo y otros utilizan conjuntos de objetos-cadena, es decir, lenguajes en el sentido usual. Un multiconjunto (un conjunto con multiplicidades asociadas a sus elementos) se corresponde de una manera directa con una cadena: basta ignorar el orden de los símbolos de ésta. Tal cosa es precisamente lo que hace la aplicación de Parikh, tan antigua en la teoría de lenguajes formales.

Así pues, muchas de las mencionadas investigaciones en la computación con ADN y en la computación con membranas tienen el característico aroma de la teoría de lenguajes formales (y de la teoría de autómatas). Hay dos razones para ello: (1) La estructura del ADN y el procesamiento del ADN, tanto *in vivo* como *in vitro*, son claramente cuestiones de tipo sintáctico (cfr. el apartado siguiente): el procesamiento de multiconjuntos está relacionado con el procesamiento de lenguajes del modo apuntado anteriormente, por lo que estas orientaciones están bastante próximas en espíritu al estudio de las lenguas naturales (ya se hizo notar esto hace muchos años: cfr., a título de ejemplo, [23]). (2) La jerarquía de gramáticas de Chomsky, equivalente a las correspondientes clases de autómatas, ofrece un abanico de mecanismos de computación (casos particulares de máquinas de Turing) que se conocen bien, por lo que constituyen una herramienta adecuada de comprobación para cualquier nuevo mecanismo de computación. En este contexto, por un lado, diversos conceptos, técnicas y resultados "clásicos"de la teoría de lenguajes formales se han aplicado (algunas veces inesperadamente) a la computación con ADN y, por otro, se han construido muchos modelos de computación con una obvia estructura "gramatical", a los que cabe considerar, por tanto, como nuevas áreas de la teoría de lenguajes formales originadas en la computación molecular. Ilustraremos la primera idea con las caracterizaciones de los lenguajes recursivamente enumerables en términos de conjuntos de igualdades y de intercalación gemela, la (prueba de la) forma normal de Geffert y la forma normal binaria para las gramáticas matriciales, mientras que la segunda cuestión se ejemplificará con los llamados sistemas adherentes, los autómatas de Watson-Crick (la réplica para el caso del ADN de los clásicos autómatas finitos de dos cabezas) y los sistemas de empalme. Apuntaremos también nuevos problemas sobre las nociones y resultados clásicos.

Naturalmente, nuestra presentación es bastante preliminar. Queremos sólo convencer al lector de que la teoría de lenguajes formales está íntimamente relacionada con la computación molecular, de que la cooperación entre ambos campos es mutuamente beneficiosa y de que de esta colaboración podría obtener provecho también incluso la lingüística.

Admítasenos la siguiente especulación. Los modelos teóricos de la computación, sean gramáticas de Chomsky, máquinas de Turing y sus variantes, sistemas de Post o de Thue, algoritmos normales de Markov, etc., se basan esencialmente en la reescritura, o sea, en sustituir una subcadena de una cadena por otra cadena. Sin embargo, la naturaleza parece "computar"empleando principalmente otras clases de operaciones, como el anillamiento (construir una secuencia doble de símbolos a partir de secuencias simples), la recombinación de moléculas (cortar dos moléculas y enganchar el prefijo de una de ellas con el sufijo de la otra), el mecanismo de inserción-supresión y otras, operaciones todas ellas que a primera vista parecen muy diferentes de la reescritura a la que estamos acostumbrados. Además, hay que tener en cuenta que los mecanismos de computación basados en tales operaciones son computacionalmente completos, esto es, capaces de caracterizar los lenguajes recursivamente enumerables.

Surgen, entonces, algunas preguntas: ¿cuál es el modo más "natural"de computar?, ¿por qué no reconstruir la teoría de la computabilidad sobre la base de estos mecanismos que no utilizan la reescritura?, ¿tiene esto algo que ver con la manera como el cerebro se supone que funciona? No proseguimos aquí en esta dirección: cfr. [25] para una ulterior discusión sobre esta materia de un evidente interés interdisciplinario.



Watson (arriba) y Crick

El ADN: Su Estructura y Operaciones

El escenario general de la búsqueda de herramientas de computación molecular es el siguiente: buscamos primero *apoyos empíricos*, de tipo bioquímico (por ejemplo, la molécula de ADN); en muchos casos, éstos se corresponden con nuevas *estructuras de datos* (en el caso del ADN, la doble secuencia de símbolos, que están emparejados de acuerdo con una *relación de complementariedad* bien especificada); si también tenemos *operaciones* con estas estructuras de datos, entonces podemos definir una *computación* como una sucesión de transiciones basadas en tales operaciones. Cómo definir una computación, su *input* de entrada y su *output* de salida, su eficiencia, etc. son cuestiones técnicas.

Para empezar, consideramos la posibilidad de utilizar moléculas de ADN como vehículo de computaciones, de aquí que brevemente presentemos la estructura y las operaciones de las mismas, obviando al mismo tiempo todos los detalles bioquímicos que no son de interés para los parágrafos siguientes.

Para nuestro propósito, una molécula de ADN es una secuencia doble compuesta de ocurrencias de cuatro letras: A (que denota el nucleótido denominado adenina), C (citosina), G (guanina) y T (timina). En cada una de las dos tiras puede haber cualquier sucesión de nucleótidos, pero los nucleótidos situados en los lugares equivalentes de las dos tiras solamente pueden corresponder a los pares AT, TA, CG y GC. Se dice que A y T, por una parte, y C y G, por otra, son *complementarios en el sentido de Watson-Crick*. La Figura 1 representa una molécula de ADN. Un detalle importante es que sus dos tiras tienen *direccionalidad*: están marcadas en sus extremos mediante $5', 3'$, tal

como aparece en la figura (el significado químico de $5', 3'$ no importa ahora); en general, las indicaciones $5', 3'$ no se especifican, pero siempre se supone que ambas tiras están orientadas como en la figura, con $5'$ en la esquina superior izquierda y las dos tiras con direccionalidades opuestas:

$$\begin{array}{l} 5'\text{– A C C T G T A T G C –}3' \\ 3'\text{– T G G A C A T A C G –}5' \end{array}$$

Figura 1. Ejemplo de una molécula de ADN.

Ésta es una molécula *completa*. También es posible tener moléculas como la de la Figura 2, con *extremos adherentes*, tiras simples colocadas en los extremos de la molécula:

$$\begin{array}{l} \text{A C C T G G T T A A} \\ \qquad\qquad\text{C C A A T T A T A C G} \end{array}$$

Figura 2. Una molécula de ADN con extremos adherentes.

Conocemos muchas operaciones con moléculas de ADN en células vivas que se pueden controlar bien en un laboratorio. En particular, ya es fácil *escribir* y *leer* secuencias de ADN, así como *multiplicarlas* (por la PCR = Reacción en Cadena de la Polimerasa, en n pasos se pueden obtener 2^n copias de una molécula dada, lo que es muy útil en la computación con ADN). Si calentamos una solución de ADN, las dos tiras se separan: esta operación se conoce como *desnaturalización*. Inversamente, al enfriar una solución en la que hay tiras simples de nucleótidos, se forman moléculas de doble tira, emparejándose siempre nucleótidos que son complementarios en el sentido de Watson-Crick y en tiras de orientaciones opuestas: ésta es la operación conocida como *anillamiento*. Por supuesto, cuando las tiras simples que hay en la solución no encajan completamente, se obtienen por anillamiento moléculas de ADN con extremos adherentes: la Figura 3 representa esa situación:

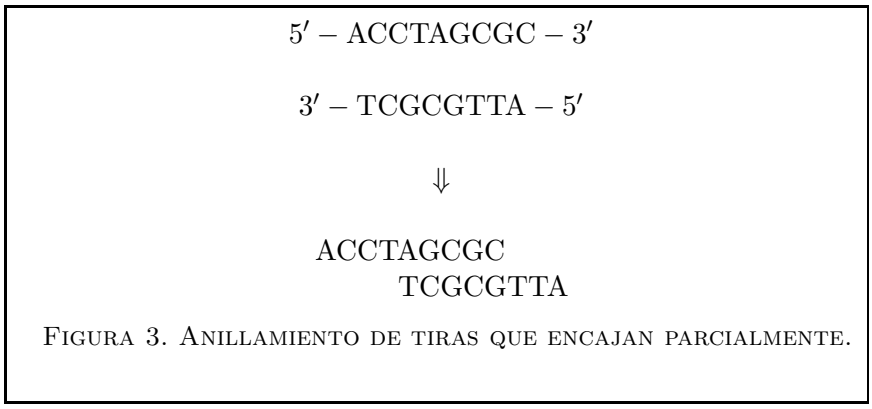


Figura 3. Anillamiento de tiras que encajan parcialmente.

Otra operación muy importante con moléculas de ADN es la de *empalme*, que, de hecho, se compone de dos operaciones simples: primero se cortan moléculas por medio de enzimas restrictivas, y luego se pegan los fragmentos obtenidos de esta forma si sus extremos adherentes encajan.

Vamos a introducir la operación de empalme a través de un ejemplo. Consideremos las siguientes dos moléculas de ADN:

así como las enzimas restrictivas *Taq*I y *Sci*NI, que reconocen, respectivamente, los siguientes patrones:

Ya hemos aludido a la manera de cortar las moléculas de ADN. Cuando actuamos sobre las dos moléculas mencionadas, estas enzimas darán lugar a los cuatro fragmentos siguientes:

$$\begin{array}{rr} 5' - \text{CCCCCT} & \text{CGACCCCC} - 3' \\ 3' - \text{GGGGGAGC} & \text{TGGGGG} - 5' \end{array}$$

$$\begin{array}{rr} 5' - \text{AAAAAG} & \text{CGCAAAAA} - 3' \\ 3' - \text{TTTTTCGC} & \text{GTTTTT} - 5' \end{array}$$

Como tenemos extremos adherentes idénticos, los cuatro fragmentos se pueden unir, bien reconstituyendo las moléculas iniciales o produciendo nuevas moléculas por recombinación. En este último caso, obtenemos las nuevas moléculas siguientes:

$$5' - \text{CCCCCTCGCAAAAA} - 3'$$
$$3' - \text{GGGGGAGCGTTTTT} - 5'$$

$$5' - \text{AAAAAGCGACCCCC} - 3'$$
$$3' - \text{TTTTTCGCTGGGGG} - 5'$$

El empalme fue formalizado ya en 1987 por T. Head y es la operación básica de uno de los modelos más desarrollados de la computación del ADN: los *sistemas H*.

Muchos detalles bioquímicos sobre la estructura del ADN y sus operaciones (incluyendo otras diferentes de las arriba mencionadas) se pueden encontrar en [34].

El Experimento de Adleman

Tal como hemos dicho en la Introducción, el primer experimento con éxito de computación con ADN fue el de L. Adleman en 1994, [1]. Dada su importan-

cia para la (historia de la) computación molecular, vamos a describirlo brevemente.

El experimento de Adleman resuelve el llamado *Problema del Camino Hamiltoniano* (Hamiltonian Path Problem) en un grafo orientado. Consideramos la siguiente formulación del problema. Sea G un grafo orientado con sus nodos de entrada y salida denotados, respectivamente, mediante v_{in} y v_{out}. Un camino de v_{in} a v_{out} se denomina *hamiltoniano* si toca cada nodo exactamente una vez.

Por ejemplo, el grafo representado en la Figura 4 tiene como nodo de entrada el 0 y como nodo de salida el 6. El camino que consta de las flechas orientadas $0 \longrightarrow 1$, $1 \longrightarrow 2$, $2 \longrightarrow 3$, $3 \longrightarrow 4$, $4 \longrightarrow 5$, $5 \longrightarrow 6$ es hamiltoniano.

L. Adleman

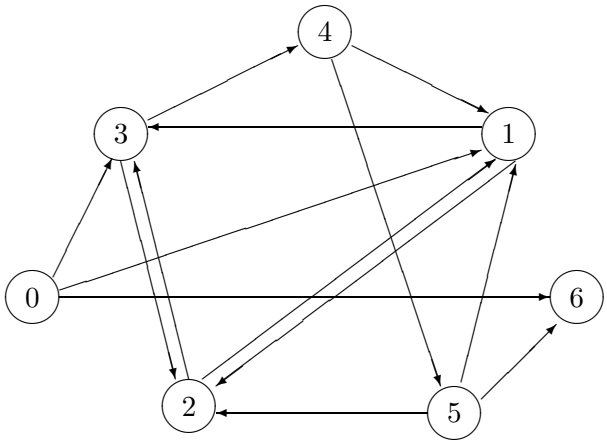


Figura 4: El grafo del experimento de Adleman.

El Problema del Camino Hamiltoniano (HPP) consiste en decidir si un grafo arbitrariamente determinado contiene un camino hamiltoniano. Es evidente que HPP puede ser resuelto mediante una búsqueda exhaustiva, y no se conoce ningún algoritmo que sea esencialmente mejor que la búsqueda exhaustiva. Más exactamente, se sabe que HPP es un problema NP-completo, lo cual significa que es improbable llegar a obtener para él un algoritmo eficiente (esto es, que funcione en tiempo polinómico). En su experimento, Adleman solucionó el HPP del ejemplo arriba mencionado, un pequeño grafo estándar en todos los aspectos, en tiempo *lineal*. No obstante, la solución hallada es, al menos en principio, aplicable también a grafos más grandes, a causa del paralelismo masivo que se obtiene al manipular ADN.

La solución de Adleman se basa en el siguiente algoritmo no determinista para resolver HPP:

> *Entrada*: Un grafo orientado G con n nodos, entre los cuales se encuentran los nodos v_{in} y v_{out}.
>
> *Paso 1:* Generar caminos en G aleatoriamente en grandes cantidades.
>
> *Paso 2:* Rechazar todos los caminos que no empiezan en v_{in} y terminan en v_{out}.
>
> *Paso 3:* Rechazar todos los caminos que no tocan exactamente n nodos.
>
> *Paso 4:* Para cada uno de los n nodos v, rechazar todos los caminos que no tocan v.
>
> *Salida:* "Sí" si subsiste algún camino, "no" en cualquier otro caso.

Esencialmente este algoritmo realiza una búsqueda exhaustiva, y se implementó en un medio bioquímico como sigue.

A cada nodo i del grafo se le asocia una tira simple de ADN de longitud 20 (es decir, compuesta de 20 nucleótidos): denotémosla mediante $s_i, 0 \leq i \leq 6$. Por ejemplo, para $i = 2, 3, 4$, Adleman empleó los siguientes oligonucleótidos[4] de longitud 20:

$$s_2 = \text{TATCGGATCGGTATATCCGA},$$
$$s_3 = \text{GCTATTCGAGCTTAAAGCTA},$$
$$s_4 = \text{GGCTAGGTACCAGCATGCTT}.$$

En cuanto a su orientación, todos estos oligonucleótidos se escriben de $5'$ a $3'$.

Considérese una función h que hace corresponder cada una de las bases de ADN con su complementario en el sentido de Watson–Crick:

$$h(\text{A}) = \text{T},\ h(\text{T}) = \text{A},\ h(\text{C}) = \text{G},\ h(\text{G}) = \text{C}.$$

[4]Las secuencias cortas de nucleótidos se llaman oligonucleótidos, abreviadamente oligos.

Para tiras de ADN, h se aplica letra a letra:

$$h(\text{CATTAG}) = \text{GTAATC}.$$

Así, h produce el complemento de Watson–Crick de una tira (h cambia la orientación de este modo: si la tira original se escribe de $5'$ a $3'$, entonces el complemento de Watson-Crick se escribirá de $3'$ a $5'$). La aplicación h es un *morfismo* de acuerdo con la teoría de lenguajes formales. Por ejemplo,

$$h(s_2) = \text{ATAGCCTAGCCATATAGGCT},$$
$$h(s_3) = \text{CGATAAGCTCGAATTTCGAT}.$$

Descompongamos ahora cada $s_i, 0 \leq i \leq 6$, en dos tiras, cada una de ellas de longitud 10: $s_i = s_i' s_i''$. Así, s_i' (respectivamente s_i'') puede ser vista como la primera (respectivamente la segunda) mitad de s_i. Una flecha desde el nodo i hasta el nodo j, supuesto que exista en el grafo G, se codifica como $h(s_i'' s_j')$. Así pues, también una flecha se codificará como un oligonucleótido de longitud 20, que se obtiene como el complemento de Watson–Crick de las mitades segunda y primera de los oligonucleótidos que codifican los nodos que toca la flecha. Debajo figuran las codificaciones de tres flechas concretas:

$$e_{2\to 3} = \text{CATATAGGCTCGATAAGCTC},$$
$$e_{3\to 2} = \text{GAATTTCGATATAGCCTAGC},$$
$$e_{3\to 4} = \text{GAATTTCGATCCGATCCATG}.$$

Una observación importante es que esta construcción preserva la orientación de las flechas: $e_{2\to 3}$ y $e_{3\to 2}$ son enteramente diferentes.

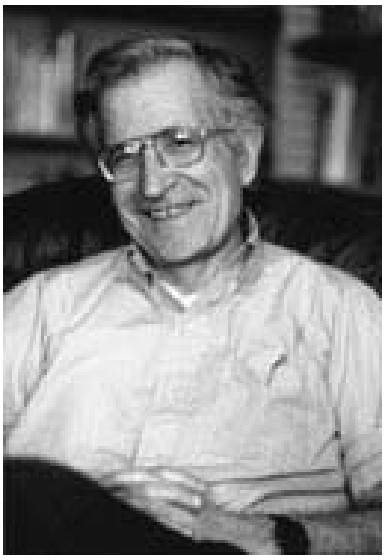
Entonces, Adleman procedió de la manera que sigue. Para cada nodo i y para cada flecha $i \longrightarrow j$ del grafo, se mezclaron grandes cantidades de oligonucleótidos s_i y $e_{i\to j}$ conjuntamente en una única reacción de ligación. Aquí los oligonucleótidos s_i sirvieron como tablillas para juntar oligonucleótidos asociados con flechas compatibles para la ligación. En consecuencia, la reacción de ligación originó la formación de moléculas de ADN que podrían ser vistas como codificaciones de caminos aleatorios en el grafo. Es importante observar que, de este modo, "todos" los caminos del grafo se pueden "generar" (naturalmente, ello significa que se proporcionan "suficientes" oligonucleótidos iniciales y que los caminos con ciclos se "generan" solamente hasta una longitud acotada).

Luego, por procedimientos estándar de filtrado, a partir del conjunto de soluciones potenciales, Adleman selecciona moléculas de longitud 140 (ello garantiza que corresponden a caminos que pasan por 7 nodos), que codifican caminos que empiezan en el nodo 0, terminan en el nodo 6 y contienen todos los nodos 1, 2, 3, 4, 5 (debido a la longitud, sabemos de esta manera que cada nodo aparece *exactamente una vez* en el camino). Como por lo menos una molécula habrá sobrevivido a este proceso de filtrado, de ahí se sigue que el grafo contiene un camino hamiltoniano.

Todo este proceso bioquímico dura aproximadamente una semana, un tiempo enorme para un pequeño grafo para el cual el problema se puede resolver mediante un simple examen. Pero aquí la eficiencia del procedimiento no es lo importante, sino el hecho de que este experimento ha demostrado prácticamente que *podemos computar usando ADN*. El proceso no es eficiente en absoluto (por ejemplo, en lo que se refiere al número de moléculas necesarias de ADN), pero la conclusión es notablemente importante: el paralelismo masivo de las reacciones bioquímicas con ADN se puede utilizar para resolver (en principio) en tiempo lineal problemas intratables para los ordenadores usuales (secuenciales).

Prerrequisitos de Teoría de Lenguajes Formales

Sólo nos referiremos a algunas nociones de la teoría de lenguajes formales que se usarán en los apartados que siguen.



N. Chomsky

Dado un *alfabeto* (un conjunto finito no vacío de símbolos abstractos) V, denotamos con V^* el conjunto de todas las cadenas de elementos de V, incluida la cadena vacía. En términos algebraicos, V^* es el monoide libre generado por V con la operación de concatenación; el elemento identidad del monoide es la cadena vacía, λ. La longitud de $x \in V^*$ se representa mediante $|x|$, en tanto que $|x|_a$ es el número de ocurrencias en x del símbolo $a \in V$. Si $V = \{a_1, \ldots, a_n\}$ (el orden es importante), entonces la *aplicación de Parikh* asociada a V es $\Psi_V : V^* \longrightarrow \mathbf{N}^n$, definida por $\Psi_V(x) = (|x|_{a_1}, \ldots, |x|_{a_n})$ para cada $x \in V^*$. Todo subconjunto de V^* es un *lenguaje* sobre V. La aplicación de Parikh se extiende de una manera natural de cadenas a lenguajes. La *cadena inversa* (imagen especular) de $x \in V^*$ se denota con $mi(x)$.

Una *gramática de Chomsky* es una estructura $G = (N, T, S, P)$ en la que N, T son alfabetos disjuntos, $S \in N$ y P es un subconjunto finito de $(N \cup T)^* N (N \cup T) \times (N \cup T)^*$. Los elementos de N se denominan símbolos *no terminales*, los de T símbolos *terminales*, S es el *axioma* de la gramática y P es el conjunto de *reglas de reescritura* (también llamadas *producciones*). Las reglas suelen escribirse $u \to v$.

Dadas $x, y \in V^*$, escribimos $x \Longrightarrow y$ si y sólo si $x = x_1 u x_2, y = x_1 v x_2$, $x_1, x_2 \in (N \cup T)^*$ y $u \to v \in P$. Decimos que y *se deriva directamente* de x en

G. La clausura reflexiva y transitiva de esta relación se representa como $\Longrightarrow^*$. El lenguaje generado por G se define como $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$.

Una gramática como ésta se dice que es de *tipo 0*. Si para todas las reglas de P es cierto que $|u| \leq |v|$, entonces la gramática es *monótona* o *dependiente del contexto*. Si $u \in N$, la gramática es *independiente del contexto*; si además v contiene como máximo un símbolo no terminal, se dice que es *lineal*. Si todas las reglas son de alguna de las formas $A \to aB$, $A \to a$ o $A \to \lambda$, con $A, B \in N, a \in T$, entonces la gramática se llama *regular*. Las familias de lenguajes generados por gramáticas regulares, lineales, independientes del contexto, dependientes del contexto y arbitrarias se representan, respectivamente, mediante *REG, LIN, CF, CS, RE*. Sabemos que:

$$FIN \subset REG \subset LIN \subset CF \subset CS \subset RE$$

son inclusiones estrictas (FIN denota la familia de lenguajes finitos). Esto se conoce con el nombre de *jerarquía de Chomsky*.

He aquí dos ejemplos de gramáticas de Chomsky. La primera es independiente del contexto y la segunda dependiente del contexto:

$$G_1 = (\{S\}, \{[\,,]\}, S, \{S \to SS,\ S \to [\ S\ ],\ S \to \lambda\}),$$
$$G_2 = (\{S, B\}, \{a, b, c\}, S, \{S \to abc,\ S \to aSBc,\ cB \to Bc,\ bB \to bb\}).$$

Dejamos al lector la tarea de verificar que G_1 genera el conjunto de todas las secuencias de paréntesis correctamente encajados [and] (incluida la cadena vacía, λ), mientras que para la segunda gramática tenemos:

$$L(G_2) = \{a^n b^n c^n \mid n \geq 1\}.$$

(Este es un lenguaje estándar dependiente del contexto).

Todas las familias integradas en la jerarquía de Chomsky, salvo FIN, pueden ser caracterizadas también con autómatas. Un autómata es un dispositivo que *reconoce/acepta* cadenas, y por consiguiente funciona en la dirección opuesta a como lo hace una gramática, que *genera* cadenas. Solamente recordaremos aquí la noción de autómata finito.

Un *autómata finito* es una estructura $A = (K, T, s_0, F, \delta)$, en la que K, T son alfabetos disjuntos (el conjunto de estados y el alfabeto de entrada, respectivamente), $s_0 \in K$ (el estado inicial), $F \subseteq K$ (el conjunto de estados finales) y $\delta : K \times T \longrightarrow 2^K$ (la función de transición). Para $s, s' \in K, a \in T, x \in T^*$, escribimos $(s, ax) \vdash (s', x)$ si $s' \in \delta(s, a)$. Por definición, $(s, \lambda) \vdash (s, \lambda)$. Si $\vdash^*$ indica la clausura reflexiva y transitiva de la relación $\vdash$, el lenguaje reconocido por A se define como:

$$L(A) = \{w \in T^* \mid (s_0, w) \vdash^* (s_f, \lambda), s_f \in F\}.$$

Si $card(\delta(s, a)) \leq 1$ para todo $a \in K, a \in V$, entonces se dice que el autómata A es *determinista*.

Sabemos que los autómatas finitos deterministas y los no deterministas son equivalentes y que reconocen exactamente la clase de los lenguajes regulares. Podemos imaginar un autómata finito como un mecanismo formado por una cinta infinita compuesta de casillas en las cuales se escribe la cadena de entrada (un símbolo en cada casilla), una memoria capaz de almacenar un estado de entre un conjunto finito de estados, y una cabeza lectora que recorre la cinta de izquierda a derecha bajo el control del estado que hay en la memoria. La computación empieza en el estado inicial, con la cabeza lectora situada en la primera casilla de la cinta, en la que está escrito el primer símbolo (el que hay más a la izquierda) de la cadena de entrada. En cada movimiento, dependiendo del símbolo leído y del estado actual, se desplaza la cabeza una casilla hacia la derecha y cambia el estado. Si la cadena ha sido recorrida por completo y se llega a un estado final, decimos que se acepta la cadena.

He aquí un ejemplo de un autómata finito:

$$\begin{aligned} A &= (\{s_0, s_f\}, \{0,1\}, s_0, \{s_f\}, \delta), \\ & \\ & \delta(s_0, 0) = \{s_0\}, \\ & \delta(s_0, 1) = \{s_f\}, \\ & \delta(s_f, 0) = \{s_f\}, \\ & \delta(s_f, 1) = \{s_0\}. \end{aligned}$$

El lector puede ver fácilmente que este autómata comprueba si una secuencia de 0's y 1's contiene un número par o impar de ocurrencias de 1, y sólo en este último caso termina la lectura de la cadena en el estado s_f, aceptándola. O sea:

$$L(A) = \{w \in \{0,1\}^* \mid |w|_1 \text{ es un número impar}\}.$$

Una *máquina de Turing* puede ser vista como un autómata finito que tiene además la posibilidad de mover la cabeza tanto a la derecha como a la izquierda y que puede, asimismo, sustituir (sobreescribir) el símbolo leído. Una computación tendrá éxito sólo si se detiene en un estado final, no importa dónde se encuentre en ese momento la cabeza lectora/escritora.

Las máquinas de Turing reconocen la familia RE de lenguajes generados por gramáticas de Chomsky de tipo 0. Según la tesis de Turing-Church[5], ésta es la clase más grande de lenguajes que puede ser algorítmicamente computada

[5]La tesis, formulada en los años cuarenta del siglo XX, ha sido sistemáticamente "confirmada" con respecto a todos los modelos de computabilidad algorítmica, en el sentido de que se ha probado que las máquinas de Turing pueden simular todos los modelos de computación introducidos hasta el presente: algoritmos de Markov, sistemas de Thue y de

(enumerada). De alguna manera, la capacidad de los autómatas finitos y la de las máquinas de Turing definen los dos límites de la computabilidad, el más bajo (y no trivial) y el más alto, respectivamente.

La Capacidad Intrínseca del ADN

En este apartado nos referiremos a una conexión notablemente sorprendente entre la estructura de la molécula de ADN y la computabilidad (en el nivel de las máquinas de Turing), establecida a través de algunos resultados bastante antiguos de la teoría de lenguajes formales, como son la caracterización de los lenguajes recursivamente enumerables por medio de conjuntos de igualdades de morfismos y de lenguajes de intercalación gemela.

Para dos morfismos $h_1, h_2 : V^* \longrightarrow U^*$, el conjunto:

$$EQ(h_1, h_2) = \{w \in V^* \mid h_1(w) = h_2(w)\}$$

se denomina *conjunto igualdad* de h_1, h_2.

Teorema 1 *Todo lenguaje recursivamente enumerable $L \subseteq T^*$ se puede escribir como $L = pr_T(EQ(h_1, h_2) \cap R)$, donde h_1, h_2 son dos morfismos, R es un lenguaje regular y pr_T es la proyección asociada al alfabeto T.*

A. Turing

El lector puede encontrar las demostraciones en [4], [42] y [43]. Una variante se probó en [18]:

Teorema 2 *Para cada lenguaje recursivamente enumerable $L \subseteq T^*$, existen dos morfismos λ-independientes h_1, h_2, un lenguaje regular R y una proyección pr_T tales que $L = pr_T(h_1(EQ(h_1,\ h_2)) \cap R)$.*

Post, cálculo lambda, gramáticas de Chomsky, etc. Por supuesto, la pregunta es bastante resbaladiza: ¿qué es esa "computabilidad algorítmica" si no disponemos de una definición matemática de este concepto? En particular, se han imaginado diversos modos de computar "más allá de Turing", generalmente modificando la definición: por ejemplo, si consideramos computaciones infinitas o ingredientes no discretos (continuos: espacio o tiempo), entonces podemos "computar"funciones que no son Turing computables. En este artículo, entendemos la computabilidad en el sentido restringido de Turing, como funciones de computación (de resolución de problemas, de generación o reconocimiento de lenguajes) en un tiempo finito discreto mediante un algoritmo que puede ser simulado por una máquina de Turing.

Obsérvese la diferencia existente entre las representaciones de los Teoremas 1 y 2: en el primer caso el lenguaje L se obtiene como proyección de la intersección del conjunto de igualdades con un lenguaje regular, mientras que en el segundo caso L es la proyección de la intersección de un lenguaje regular con la imagen del conjunto de igualdades por uno de los morfismos que definen el conjunto de igualdades.

Una importante consecuencia del Teorema 1 es la siguiente. Considérese un alfabeto V y su variante $\overline{V} = \{\bar{a} \mid a \in V\}$. El lenguaje:

$$TS_V = \bigcup_{x \in V^*} (x \sqcup\!\sqcup \bar{x})$$

recibe el nombre de lenguaje de *intercalación gemela* sobre V. (Para una cadena $x \in V^*$, $\bar{x}$ denota la cadena que se obtiene sustituyendo cada símbolo de x por su variante con barra, y $x \sqcup\!\sqcup \bar{x}$ es el conjunto de cadenas construidas intercalando los símbolos de x con los de $\bar{x}$ de todas las maneras posibles).

El siguiente resultado fue demostrado ya en [10]:

Teorema 3 *Todo lenguaje recursivamente enumerable $L \subseteq T^*$ se puede escribir como $L = pr_T(TS_V \cap R)$, donde V es un alfabeto y R es un lenguaje regular.*

En esta representación, el lenguaje TS_V depende del lenguaje L. Esto se puede evitar de la siguiente forma. Tómese una codificación $f : V \longrightarrow \{0,1\}^*$, por ejemplo $f(a_i) = 01^i0$, donde a_i es el símbolo de V que ocupa la posición i en un orden especificado. El lenguaje $f(R)$ es regular. Una *gsm* (un transductor secuencial finito) puede simular la intersección con un lenguaje regular, la proyección pr_T así como la decodificación de los elementos de $f(TS_V)$. En consecuencia, se obtiene:

Corolario 1 *Para cada lenguaje recursivamente enumerable L, existe una gsm g_L tal que $L = g_L(TS_{\{0,1\}})$.*

Por tanto, todo lenguaje recursivamente enumerable se puede obtener por medio de un transductor secuencial partiendo de un lenguaje único, $TS_{\{0,1\}}$ (en terminología de AFL = familias abstractas de lenguajes, $TS_{\{0,1\}}$ sería considerado un *generador* de RE). También se puede demostrar que este transductor puede ser determinista.

Dado que un transductor secuencial finito es un tipo muy particular de máquina, podemos inferir que la principal capacidad computacional reside en el lenguaje de intercalación gemela. De modo bastante sorprendente – y significativo desde la perspectiva de la computación molecular – existe una conexión directa entre $TS_{\{0,1\}}$ y el ADN, apuntada por primera vez en [39] (y también discutida en [34]).

Considérese la codificación de los cuatro nucleótidos que aparecen en las dos tiras de una molécula de ADN según sugiere la tabla siguiente:

	tira superior	tira inferior
A, T	0	$\bar{0}$
C, G	1	$\bar{1}$

Esto es, ambos nucleótidos A y T se identifican con 0, sin barra cuando aparecen en la tira superior y con barra cuando aparecen en la tira inferior; por su parte, los nucleótidos C y G se identifican con 1 en la tira superior y con $\bar{1}$ en la tira inferior.

Consideremos ahora una molécula de doble tira, por ejemplo la de la Figura 1, e imaginemos el siguiente escenario: dos "insectos"inteligentes se hallan situados en los extremos izquierdos de las dos tiras de esta molécula y se les pide que "lean"las tiras, de izquierda a derecha, paso a paso, a una determinada velocidad arbitraria cada uno de ellos, y que nos digan con qué nucleótido se encuentran de acuerdo con la codificación especificada más arriba. Es obvio que lo que obtendremos será una cadena de símbolos $0, 1, \bar{0}, \bar{1}$, por tanto una cadena de $TS_{\{0,1\}}$. Lo que es verdaderamente importante es que de esta forma podemos obtener *todas* las cadenas de $TS_{\{0,1\}}$: basta considerar *todas* las moléculas (secuencias completas de dos tiras) y *todas* las posibilidades de leerlas tal como se ha precisado antes. Alcanzamos idéntico resultado si usamos moléculas que contengan en la tira superior solamente nucleótidos de cualquiera de los pares:

(A, C), (A, G), (T, C), (T, G).

A la inversa, dada una cadena x de $TS_{\{0,1\}}$, se puede encontrar fácilmente una molécula de ADN cuya lectura conforme al procedimiento anterior dé exactamente la cadena x. Consecuentemente, podemos escribir de modo metafórico:

$$ADN \equiv TS_{\{0,1\}},$$

y, metafóricamente también, podemos reemplazar el lenguaje de intercalación gemela en el Corolario 1 por ADN.

Cabe afirmar, pues, que *todas las computaciones posibles se encuentran codificadas en el ADN y podemos recuperarlas mediante un transductor secuencial finito determinista y dos "insectos" inteligentes como los empleados arriba.*

Hay aquí un aspecto en el cual la argumentación anterior resulta vulnerable: los dos "insectos"están situados en el extremo izquierdo de las dos tiras de una molécula, aunque sabemos (cfr. de nuevo la Figura 1) que las dos tiras están orientadas en direcciones opuestas. Tomemos en consideración esta direccionalidad opuesta y coloquemos a los "insectos"– digamos – en los extremos de ambas tiras rotulados con $5'$. Satisfactoriamente podemos afirmar que la conclusión será la misma, debido a la robustez de la caracterización de los lenguajes recursivamente enumerables por medio de los lenguajes de intercalación gemela.

Más concretamente, el resultado del Corolario 1 es válido también para una variante especular del lenguaje de intercalación gemela.

Dado un alfabeto V, considérese el lenguaje:

$$RTS_V = \bigcup_{x \in V^*} (x \sqcup\!\sqcup\, mi(\bar{x})).$$

Este es el lenguaje de *intercalación gemela inversa* asociado a V.

La siguiente versión del Corolario 1 fue demostrada en [9]:

Teorema 4 *Para cada lenguaje recursivamente enumerable L, existe una gsm determinista g_L tal que $L = g_L(RTS_{\{0,1\}})$.*

Es evidente que, leyendo la tira superior de una molécula de izquierda a derecha y la inferior de derecha a izquierda, a velocidades no correlacionadas y no deterministas, y usando la codificación de nucleótidos anterior, llegamos a cadenas de $RTS_{\{0,1\}}$. Todas esas cadenas se pueden obtener utilizando todas las moléculas y todas las lecturas posibles. Otra vez, cabe identificar el lenguaje $RTS_{\{0,1\}}$ con el ADN y observar que caracteriza la familia de los lenguajes recursivamente enumerables, módulo una gsm determinista y una cierta codificación de nucleótidos.

Aquí surge ahora un problema matemático natural, que también conduce a especulaciones biológicas interesantes: ¿son necesarios los cuatro símbolos $0, 1, \bar{0}, \bar{1}$ (los cuatro nucleótidos) para llegar a las caracterizaciones de RE que hemos visto? De una manera bastante sorprendente, comprobamos que la respuesta es negativa.

Consideremos la codificación $c : \{0,1\}^* \longrightarrow \{\bar{0}, 1\}^*$ definida por $c(0) = \bar{0}$ y $c(1) = 1$, y consideremos el lenguaje (denominado lenguaje de *intercalación semigemela* sobre $\{0,1\}$):

$$STS_{\{0,1\}} = \bigcup_{x \in \{0,1\}^*} (x \sqcup\!\sqcup\, c(x)).$$

Observe el lector que empleamos sólo tres símbolos, dado que el símbolo 1 es su propio complemento. Aun así, todavía tenemos un resultado como el del Teorema 1.

Teorema 5 *Para cada lenguaje recursivamente enumerable L, existe una gsm g_L tal que $L = g_L(STS_{\{0,1\}})$.*

Otra réplica del Teorema 4 se formula en [27] para el lenguaje de *intercalación semigemela inversa*:

$$RSTS_{\{0,1\}} = \bigcup_{x \in \{0,1\}^*} (x \sqcup\!\sqcup\, c(mi(x))).$$

Las demostraciones se pueden encontrar en [27] (y se recogen en [34]), donde asimismo se plantea el problema siguiente: ¿qué sucede con variantes aún más débiles? Un candidato es el lenguaje de *autointercalación* sobre un alfabeto V, que se denota con FS_V y se define por:

$$FS_V = \bigcup_{x \in V^*} (x \sqcup\!\sqcup\, x).$$

(Cada símbolo es su propio complemento, de tal manera que nos limitamos a intercalar cadenas consigo mismas).

Es un problema no resuelto si podemos caracterizar o no la familia de los lenguajes recursivamente enumerables usando $FS_{\{0,1\}}$. En [27] solamente se probó que los lenguajes FS_V, para cualquier alfabeto V que contenga al menos dos símbolos, así como los lenguajes de autointercalación inversa sobre alfabetos V que contengan al menos tres símbolos (su definición es inmediata) no son independientes del contexto.

Sistemas Adherentes y Autómatas de Watson-Crick

Aparte de su interés teórico, la conexión aludida anteriormente entre la estructura de las moléculas del ADN y el lenguaje de intercalación gemela también tiene consecuencias directas para la computación (teórica) del ADN. En efecto, la caracterización de los lenguajes recursivamente enumerables con conjuntos de igualdades sugiere considerar una clase de mecanismos generadores de lenguajes que estén basados en la operación de anillamiento, conocidos con el nombre de *sistemas adherentes*, mientras que la caracterización en términos de intercalación gemela condujo a la introducción de los llamados *autómatas de Watson-Crick*, que también se pueden considerar como una réplica de los sistemas adherentes en la teoría de autómatas y que, en el caso general, no son nada más que una forma nueva de presentar los autómatas finitos de dos cabezas que ya habían sido investigados en los años sesenta (cfr. [38]).

Aquí van a continuación sólo unos pocos detalles sobre estos mecanismos (cfr. [18], [13], y los capítulos pertinentes de [34] para las definiciones precisas y las demostraciones).

Considérese un alfabeto V y una relación $\rho \subseteq V \times V$ (de *complementariedad*) sobre V que sea simétrica. (La complementariedad de Watson-Crick es también inyectiva, pero aquí no imponemos esa propiedad). Denotemos:

$$\begin{bmatrix} V \\ V \end{bmatrix}_\rho = \left\{ \begin{bmatrix} a \\ b \end{bmatrix} \,\middle|\, a, b \in V, (a, b) \in \rho \right\},$$

$$WK_\rho(V) = \begin{bmatrix} V \\ V \end{bmatrix}_\rho^*.$$

El conjunto $WK_\rho(V)$ recibe el nombre de *dominio de Watson-Crick* asociado al alfabeto V y a la relación de complementariedad ρ. Los elementos

$\begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \end{bmatrix} \dots \begin{bmatrix} a_n \\ b_n \end{bmatrix} \in WK_\rho(V)$ también pueden escribirse como $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$, para $w_1 = a_1 a_2 \dots a_n, w_2 = b_1 b_2 \dots b_n$. Dada una secuencia (a veces decimos una *molécula*) $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \in WK_\rho(V)$, las dos cadenas que la componen, w_1, w_2, se llaman *tiras*; w_1 es la tira *superior* y w_2 es la tira *inferior*.

Además, vamos a utilizar moléculas *incompletas*, esto es, elementos del conjunto:

$$W_\rho(V) = L_\rho(V) \cup R_\rho(V) \cup LR_\rho(V),$$

con:

$$\begin{aligned} L_\rho(V) &= \left(\begin{pmatrix} \lambda \\ V^* \end{pmatrix} \cup \begin{pmatrix} V^* \\ \lambda \end{pmatrix} \right) \begin{bmatrix} V \\ V \end{bmatrix}_\rho^*, \\ R_\rho(V) &= \begin{bmatrix} V \\ V \end{bmatrix}_\rho^* \left(\begin{pmatrix} \lambda \\ V^* \end{pmatrix} \cup \begin{pmatrix} V^* \\ \lambda \end{pmatrix} \right), \\ LR_\rho(V) &= \left(\begin{pmatrix} \lambda \\ V^* \end{pmatrix} \cup \begin{pmatrix} V^* \\ \lambda \end{pmatrix} \right) \begin{bmatrix} V \\ V \end{bmatrix}_\rho^+ \left(\begin{pmatrix} \lambda \\ V^* \end{pmatrix} \cup \begin{pmatrix} V^* \\ \lambda \end{pmatrix} \right). \end{aligned}$$

Las formas posibles de los elementos de $W_\rho(V)$ se ilustran en la Figura 5. En todos los casos tenemos una secuencia doble bien formada x y brazos y, z a uno o a los dos lados de x. Estos brazos (extremos adherentes) pueden localizarse en la tira superior o en la inferior. Obsérvese que en el caso de $L_\rho(V)$ y $R_\rho(V)$ el bloque x puede ser vacío, pero en los elementos de $LR_\rho(V)$ el bloque x contiene por lo menos un elemento $\begin{bmatrix} a \\ b \end{bmatrix}$, con $(a, b) \in \rho$. A su vez, los brazos pueden ser también vacíos: lo que queda entonces es un elemento de $WK_\rho(V)$.

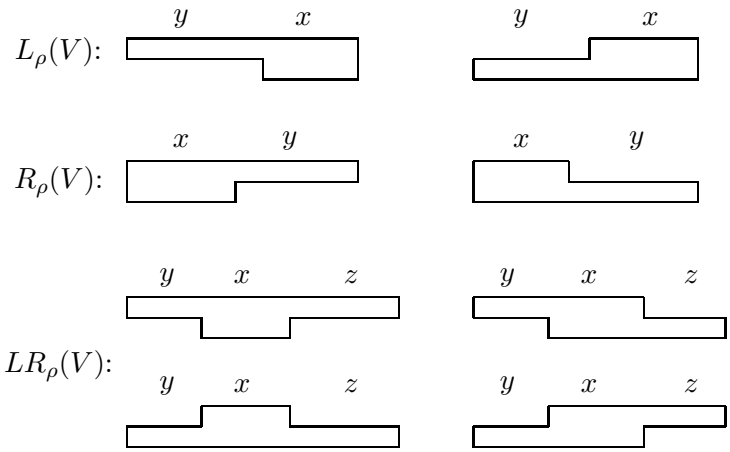


Figura 5. Formas posibles de dominós.

Entre los elementos de $W_\rho(V)$ cabe definir una operación parcial que modelice el anillamiento (seguida de un sellado): un inicio de molécula bien formada x (por tanto una secuencia con al menos una posición ocupada en ambas tiras) se puede prolongar a la derecha o a la izquierda con una molécula y, en el supuesto de que los respectivos extremos adherentes encajen, es decir, sean complementarios en las posiciones correspondientes. El resultado debería ser siempre un inicio de molécula bien construida, así pues una secuencia que no tiene lugares vacíos rodeados por símbolos de V.

Los ocho casos posibles en los que una molécula x es prolongada a la derecha con una molécula y se representan en la Figura 6. La operación se denota como $\mu(x, y)$:

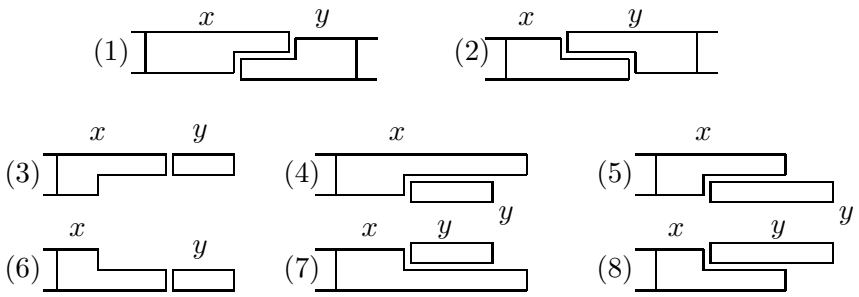


Figura 6. La operación de adherencia.

De modo simétrico, podemos definir $\mu(y, x)$, la prolongación a la izquierda de un inicio de molécula bien formada x con una secuencia y. Nótese que no necesitamos distinguir la "prolongación izquierda" de la "prolongación derecha" representándolas de diferentes maneras: en cualquiera de los casos, por lo menos uno de los términos de la operación tiene que ser un inicio de molécula bien formada, y el resultado depende enteramente del orden de las dos secuencias y de sus extremos adherentes.

De la misma forma que la reescritura es la operación subyacente a las gramáticas de Chomsky, la operación de adherencia es la que subyace a los sistemas adherentes. Introducimos aquí estos mecanismos en su forma general, o sea, como estructuras:

$$\gamma = (V, \rho, A, D),$$

donde V es un alfabeto, $\rho \subseteq V \times V$ es una relación simétrica, A es un subconjunto finito de $LR_\rho(V)$ y D es un subconjunto finito de $W_\rho(V) \times W_\rho(V)$.

Los elementos de A reciben el nombre de *axiomas*. A partir de estos axiomas y empleando los pares (u, v) de dominós de D, podemos obtener un conjunto de secuencias dobles de $WK_\rho(V)$, por tanto moléculas completas, mediante el uso de la operación μ de adherencia.

Formalmente, dados un sistema adherente $\gamma = (V, \rho, A, D)$ y dos secuencias $x, y \in LR_\rho(V)$, definimos:

$$x \Longrightarrow y \text{ ssi } y = \mu(u, \mu(x, v)), \text{ para algún } (u, v) \in D.$$

Una secuencia $x_1 \Longrightarrow x_2 \Longrightarrow \ldots \Longrightarrow x_k$, con $x_1 \in A$, se denomina una *computación* en γ. Una computación $x_1 \Longrightarrow^* x_k$ es *completa* cuando $x_1 \in A$ y $x_k \in WK_\rho(V)$ (se empieza desde un axioma y no hay ningún extremo adherente en la última secuencia).

El lenguaje generado por γ, $L(\gamma)$, consta de todas las cadenas que hay en la tira superior de una molécula a la que se llega a través de una derivación completa en γ.

Diversas variedades de sistemas adherentes son de interés. Se dice que un sistema $\gamma = (V, \rho, A, D)$ es:

- *unilateral* si para cada par $(u, v) \in D$, o bien $u = \lambda$ o bien $v = \lambda$;
- *regular* si para cada par $(u, v) \in D$, $u = \lambda$;
- *simple* si todos los pares $(u, v) \in D$ tienen o bien $u, v \in \binom{V^*}{\lambda}$ o bien $u, v \in \binom{\lambda}{V^*}$.

En los sistemas unilaterales, la prolongación a la izquierda es independiente de la prolongación a la derecha. En los sistemas regulares, solamente prolongamos las secuencias a la derecha (de ahí que los axiomas tengan que ser de la forma $x_1 x_2$, con $x_1 \in WK_\rho(V)$ y $x_2 \in \binom{V^*}{\lambda} \cup \binom{\lambda}{V^*}$). En una computación en un sistema adherente simple, añadimos símbolos a una sola de las dos tiras.

Notamos mediante ASL la familia de lenguajes de la forma $L(\gamma)$, siendo γ un sistema adherente de tipo arbitrario (SL abrevia "lenguaje adherente" y A indica que se utilizan sistemas adherentes de tipo "arbitrario"). Cuando solamente se emplean sistemas adherentes que son unilaterales, regulares, simples, simples y unilaterales, o simples y regulares, sustituimos la A delante de SL por O, R, S, SO, SR, respectivamente.

Recordamos aquí, de [34], dos resultados esenciales sobre la capacidad generativa de los sistemas adherentes (y remitimos al lector allí también para ejemplos de sistemas adherentes):

Teorema 6 *$OSL = RSL = REG$.*

Teorema 7 *Todo lenguaje $L \in RE$ se puede escribir como $L = h(L')$, donde h es una codificación débil y $L' \in SSL$.*

La situación que muestran estos dos últimos teoremas ilustra un hecho muy importante: una operación natural (el anillamiento aquí), usada de una manera libre (como en los sistemas adherentes unilaterales), nos puede llevar sólo a los lenguajes regulares; mientras que si imponemos un control sobre la misma (en nuestro caso, el emparejamiento de los dominós empleados en cada paso), la capacidad del mecanismo se incrementa espectacularmente hasta alcanzar a caracterizar los lenguajes recursivamente enumerables.

Un equivalente en forma de autómata de los sistemas adherentes fue esbozado en la forma como "leímos" las moléculas de ADN en la sección La capacidad intrínseca del ADN, cuando apuntábamos el "isomorfismo" existente entre el ADN y el lenguaje de intercalación gemela. Lo llamamos *autómata*

finito de Watson-Crick. La Figura 7 ilustra el aspecto de esta máquina, que se define formalmente como una estructura:

$$M = (K, V, \rho, s_0, F, \delta),$$

donde K, V son alfabetos disjuntos, $\rho \subseteq V \times V$ es una relación simétrica, $s_0 \in K$, $F \subseteq K$ y $\delta : K \times \binom{V^*}{V^*} \longrightarrow \mathcal{P}(K)$ es una función tal que $\delta\left(s, \binom{x_1}{x_2}\right) \neq \emptyset$ solamente para un número finito de tripletas $(s, x_1, x_2) \in K \times V^* \times V^*$.

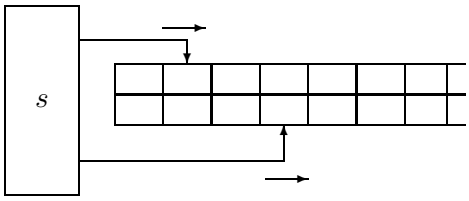


Figura 7. Un autómata finito de Watson-Crick.

Los elementos de K se denominan *estados*, V es el alfabeto (de entrada), ρ es una relación de complementariedad sobre V, s_0 es el estado inicial, F es el conjunto de estados finales y δ es la función de transición. La interpretación de $s' \in \delta(s, \binom{x_1}{x_2})$ es la siguiente: en el estado s, el autómata procesa x_1 en la tira superior y x_2 en la tira inferior de una secuencia doble, y accede al estado s'. Esto es, tenemos un autómata finito con una cinta doble, leída por dos cabezas separadamente.

Una transición en un autómata finito de Watson-Crick se puede definir como sigue: dados $\binom{x_1}{x_2}, \binom{u_1}{u_2}, \binom{w_1}{w_2} \in \binom{V^*}{V^*}$ tales que $\begin{bmatrix} x_1u_1w_1 \\ x_2u_2w_2 \end{bmatrix} \in WK_\rho(V)$ y $s, s' \in K$, escribimos:

$$\binom{x_1}{x_2} s \binom{u_1}{u_2} \binom{w_1}{w_2} \Longrightarrow \binom{x_1}{x_2} \binom{u_1}{u_2} s' \binom{w_1}{w_2}$$
$$\text{ssi } s' \in \delta\left(s, \binom{u_1}{u_2}\right).$$

Denotamos mediante $\Longrightarrow^*$ la clausura reflexiva y transitiva de la relación $\Longrightarrow$.

Al igual que en el caso de los sistemas adherentes, consideramos aquí el lenguaje reconocido por nuestro autómata como el compuesto por las cadenas

que aparecen en las tiras superiores de cintas de Watson-Crick, es decir, el lenguaje:

$$L(M) = \left\{ w_1 \in V^* \mid s_0 \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \Longrightarrow^* \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} s_f, \right.$$
$$\left. \text{con } s_f \in F, \text{ y } w_2 \in V^*, \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \in WK_\rho(V) \right\}.$$

En una descripción informal, en el estado inicial comenzamos con las dos cabezas situadas sobre los símbolos del extremo izquierdo de cada una de las tiras, y a continuación movemos las cabezas hacia la derecha de acuerdo con las transiciones que describe δ. La cadena de la tira superior resulta reconocida si las dos cabezas llegan a los símbolos del extremo derecho de las respectivas tiras y el autómata entra en un estado final. Insistimos en el hecho de que estamos tratando con moléculas completas: las longitudes de las cadenas escritas en ambas tiras de la cinta son iguales.

A partir de la construcción anterior y de la discusión con la que comenzamos, el siguiente resultado era de esperar:

Teorema 8 *Todo lenguaje recursivamente enumerable es la proyección de un lenguaje reconocido por un autómata finito de Watson-Crick.*

También podemos conseguir caracterizar RE mediante clases restringidas de autómatas finitos de Watson-Crick, por ejemplo con todos sus movimientos de la forma $s \binom{x_1}{x_2} \rightarrow \binom{x_1}{x_2} s'$ y $|x_1 x_2| = 1$ (en cada movimiento leemos un único símbolo, bien sea en la tira superior o en la inferior); con un único estado; con todos sus estados que también sean finales; etc. Para estos casos restringidos ya no es suficiente una proyección, sino que son necesarios mecanismos más poderosos (morfismos y/o intersecciones con lenguajes regulares).

Teniendo en cuenta la direccionalidad opuesta de las dos tiras de una molécula de ADN, podemos considerar una variante de autómata finito de Watson-Crick que comience a partir de una configuración como la de la Figura 8: al principio de una computación, la cabeza lectora superior se encuentra situada sobre la primera casilla de la tira superior, mientras que la cabeza inferior está sobre la casilla del extremo derecho de la tira inferior. Estos autómatas se llaman *autómatas finitos inversos de Watson-Crick*. Con ellos se consiguen caracterizaciones de los lenguajes recursivamente enumerables de la misma forma que con la variante básica.

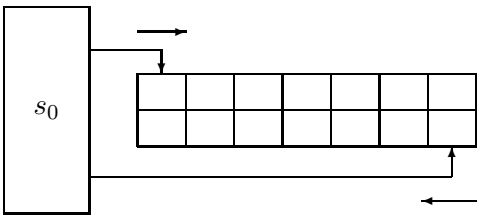


Figura 8. Un autómata finito inverso de Watson-Crick.

Es fácil demostrar que un autómata de Watson-Crick que en cada paso lea a lo sumo un símbolo de cada una de las dos tiras es equivalente a un autómata finito de dos cabezas. Es interesante resaltar que un resultado como el del Teorema 8 no se conoce para los autómatas de muchas cabezas. A su vez, como subproducto de una prueba relativa a sistemas adherentes (la demostración del Teorema 7), también podemos inferir una versión más fuerte de otro viejo resultado de la teoría de lenguajes formales, la caracterización de un lenguaje recursivamente enumerable como proyección de la intersección de dos lenguajes lineales: este resultado ya se encuentra en [2] y se puede mejorar considerando sólo lenguajes lineales *mínimos* (una gramática lineal es mínima si utiliza un único símbolo no terminal y una única regla terminal).

Otras investigaciones sobre los autómatas de Watson-Crick tienen que ver con cuestiones de complejidad descriptiva ([29]) o formas normales ([26]), temas ambos que son clásicos en la teoría de lenguajes formales.

Lenguajes de Horquilla

Vamos a describir a continuación otro vínculo muy interesante entre un antiguo resultado de la teoría de lenguajes formales y la computación con ADN que también ha conducido a nuevos temas de investigación de carácter matemático. El punto de partida es la denominada *forma normal de Geffert* para las gramáticas de Chomsky de tipo 0: para cada lenguaje $L \in RE$ podemos construir una gramática $G = (N, T, S, P)$ con $N = \{S, A, B, C, D\}$ y sus reglas de las formas siguientes:

1. $S \to uSv,\ S \to z$, con $u, v, z \in (\{A, B, C, D\} \cup T)^*$,
2. $AB \to \lambda, CD \to \lambda$.

O sea, además de dos reglas independientes del contexto del primer tipo (en cierto sentido, se podrían considerar lineales mínimas, porque el único verdadero símbolo no terminal de las mismas es S), se utilizan exactamente dos

reglas que no son independientes del contexto, las reglas de supresión del segundo tipo. La prueba de este resultado (cfr. [15]) está relacionada con las caracterizaciones de los lenguajes recursivamente enumerables por medio de conjuntos de igualdades de morfismos.

A través de un fino análisis de la demostración de [15], Yokomori observa que, de hecho, las reglas del primer tipo son o de la forma $S \to wuSv$ o de la forma $S \to uSv$, con $w \in T^*, u \in \{A, C\}^*, v \in \{B, D\}^*$, mientras que las reglas del segundo tipo tienen $x \in \{B, D\}^*$. Así, tomando A, T, C, G en lugar de A, B, C, D, podemos interpretar las reducciones $AB \to \lambda, CD \to \lambda$ como emparejamientos de Watson-Crick $(A, T), (C, G)$. Además, las derivaciones en la gramática G se pueden organizar de tal manera que primero se propone una cadena $w \in T^*$ para ser generada y se introduce una codificación de la misma así como de una derivación en una gramática de tipo 0 que la genere, y luego, usando reglas de tipo 3, se comprueba si ésta es una derivación correcta que lleva a w. Más concretamente, primero generamos una cadena de la forma wz_1z_2 empleando reglas del primer tipo de arriba, y después reducimos la cadena z_1z_2 a la cadena vacía mediante reglas del segundo tipo. En términos del ADN, ello significa que z_1 es complementario de $mi(z_2)$. Generar una molécula simple de ADN que corresponda a una cadena wz_1z_2 es algo que se puede hacer ligando moléculas construidas cuidadosamente; comprobar si una cadena de esa forma tiene un z_1 perfectamente complementario con $mi(z_2)$ (se dice que tales cadenas ponen de manifiesto una *estructura de horquilla*) también se puede hacer, en principio, en un laboratorio (aunque no conocemos ninguna implementación efectivamente realizada). Técnicamente, codificando todos los símbolos por medio de nucleótidos, la reducción de los símbolos A, B, C, D corresponde a un anillamiento completo de dos secuencias simples. Remitimos a [49] para los sutiles detalles de la transposición de la prueba formal a proceso bioquímico. Lo que obtenemos es un procedimiento para generar las cadenas de un lenguaje de tipo 0 de una forma enteramente bioquímica. Esta manera de computar es lo que se llama, desde [49], el *modelo YAC*.

En [41] se considera también una variedad de estructuras de horquilla y se presenta una solución del problema SAT (satisfacibilidad de las fórmulas proposicionales en la forma normal conjuntiva) que está basada en el ADN y se ejecuta en tiempo bioquímico polinómico (recuérdese que el SAT es un problema NP-completo). Ello sugiere considerar sistemáticamente lenguajes cuyas cadenas correspondan a diversos tipos de estructuras de horquilla. Esta investigación se inició en [35], de donde tomamos algunos detalles.

Considérese una codificación $h : V \longrightarrow V$ (extendida de manera natural a $h : V^* \longrightarrow V^*$) tal que $h(h(a)) = a$, para todo $a \in V$. Podemos tener $h(a) = a$ para algunos o para todos los símbolos $a \in V$. A esto lo denominamos *morfismo de Watson-Crick*. Si $h(a) = b$, entonces, obviamente, $h(b) = a$ y decimos que a y b son *complementarios* el uno del otro.

Cuando formamos una molécula de horquilla, es necesario que la secuencia anillada sea "suficientemente larga"a fin de asegurar la estabilidad de la

estructura, así que siempre imponemos que esta secuencia sea de longitud al menos k, para un cierto k.

Sea entonces k un número entero positivo. El *lenguaje de horquilla sin restricciones* (de grado k) se define como:

$$uH_k = \{zvwxy \mid z, v, w, x, y \in V^*, x = mi(h(v)) \text{ y } |x| \geq k\}.$$

Imponiendo restricciones sobre los lugares en que se puede producir el anillamiento, se obtienen varios sublenguajes de uH_k: cuando $z = \lambda$ en la definición anterior de uH_k, se obtiene el lenguaje bH_k; si $w = \lambda$, se añade c; y si $y = \lambda$, escribimos f delante de H_k. Así, por ejemplo, tenemos:

$$\begin{aligned} bH_k &= \{vwxy \mid v, w, x, y \in V^*, x = mi(h(v)) \text{ y } |x| \geq k\}, \\ bcH_k &= \{vxy \mid v, x, y \in V^*, x = mi(h(v)) \text{ y } |x| \geq k\}, \\ bcfH_k &= \{vx \mid v, x \in V^*, x = mi(h(v)) \text{ y } |x| \geq k\}. \end{aligned}$$

En el esquema de computación de [41] se resta un lenguaje uH_k, para algún k, de un lenguaje (regular), en tanto que en YAC [49] se intersecta un lenguaje (lineal) dado con bcH_1 (no es difícil imaginar variantes del modelo YAC, como la que utiliza el lenguaje cfH_1 en lugar de bcH_1). Ello sirve de motivación para el estudio de lenguajes de cualquiera de las dos formas αH_k y $V^* - \alpha H_k$, con $\alpha \in \{u, b, c, f, bc, bf, cf, bcf\}$. He aquí, extractados y sintetizados, algunos de los resultados de [35]:

Teorema 9 *Todos los lenguajes αH_k, para $\alpha \in \{u, b, f, c, bf\}$ y $k \geq 1$, son regulares, mientras que los restantes lenguajes, para $\alpha \in \{bc, cf, bcf\}$, son lineales no regulares. También es lineal $V^* - bcfH_k, k \geq 1$, mientras que $V^* - bcH_k$ y $V^* - cfH_k, k \geq 1$, son no independientes del contexto.*

Una investigación más cuidadosa de estos lenguajes, por ejemplo con respecto a las familias intermedias entre CF y CS, aparece formulada como problema abierto en [35].

Sistemas de Empalme

La operación bioquímica de empalme, tal como se explicó en la sección El ADN: Su Estructura y Operaciones, condujo a una operación formal con cadenas que se propuso por primera vez en [16]. Nos abstenemos de presentar en este lugar el proceso de abstracción que supone el paso del caso preciso de las moléculas de ADN al empalme de cadenas sobre un alfabeto arbitrario, e introducimos directamente el empalme en su versión general de [34].

Sean un alfabeto V y dos símbolos especiales, $\#, \$$, que no pertenecen a V. Una *regla de empalme* (sobre V) es una cadena de la forma:

$$r = u_1 \# u_2 \$ u_3 \# u_4,$$

donde $u_1, u_2, u_3, u_4 \in V^*$. Con relación a tal regla r, dados $x, y, z \in V^*$, escribimos:

$$\begin{aligned}(x, y) \vdash_r z \quad \text{ssi} \quad & x = x_1 u_1 u_2 x_2,\ y = y_1 u_3 u_4 y_2, \\ & z = x_1 u_1 u_4 y_2,\ \text{para algunos } x_1, x_2, y_1, y_2 \in V^*.\end{aligned}$$

Decimos que *empalmamos* x, y en los *puntos* $u_1 u_2, u_3 u_4$, respectivamente, y el resultado es z. Cuando queda claro por el contexto, omitimos r y escribimos solamente $\vdash$ en lugar de $\vdash_r$.

El paso de x, y a z, a través de $\vdash_r$, se puede representar tal como se muestra en la Figura 9:

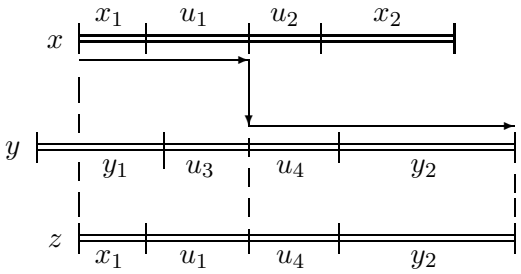


Figura 9. La operación de empalme.

A partir de esta operación, procedemos a continuación a construir un mecanismo de computación. Un *esquema* H es un par:

$$\sigma = (V, R),$$

donde V es un alfabeto y $R \subseteq V^* \# V^* \$ V^* \# V^*$ es un conjunto de reglas de empalme. Observe el lector que R puede ser infinito y que cabe plantearse cuál es su lugar en la jerarquía de Chomsky, o en cualquier otra clasificación de lenguajes. En general, si $R \in FL$, para una familia dada de lenguajes FL, entonces decimos que el esquema H σ es de *tipo* FL.

Para un cierto esquema H $\sigma = (V, R)$ y un lenguaje $L \subseteq V^*$, definimos:

$$\begin{aligned}\sigma(L) = \{z \in V^* \mid\ & (x, y) \vdash_r z, \\ & \text{con } x, y \in L, r \in R\}, \\ \sigma^0(L) = L,\ & \\ \sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)),\ & i \geq 0,\end{aligned}$$

y

$$\sigma^*(L) = \bigcup_{i \geq 0} \sigma^i(L).$$

Consecuentemente, $\sigma^*(L)$ es la clausura de L por el empalme con respecto a σ, es decir, el lenguaje más pequeño L' que contiene a L y es cerrado por el empalme con respecto a σ.

Consideremos un ejemplo simple. Sea L el lenguaje singletón $\{abba\}$ sobre el alfabeto $V = \{a, b\}$, y sea R el conjunto de dos reglas de empalme:

$$\{r_1 : a\#b\$b\#ba,\ r_2 : b\#a\$b\#ba\}.$$

Usando la primera regla, obtenemos:

$$(a|bba, ab|ba) \vdash_{r_1} aba.$$

(Con las barras verticales hemos señalado el lugar en el que cortar las dos cadenas cut). Empalmando las cadenas $aba, abba$ con la regla r_1 obtenemos la misma cadena aba. Sin embargo, empleando iterativamente la segunda regla, podemos obtener cadenas de longitud creciente:

$$(ab|a, ab|ba) \vdash_{r_2} abba,$$
$$(abb|a, ab|ba) \vdash_{r_2} abbba,$$

y, en general:

$$(ab^n|a, ab|ba) \vdash_{r_2} ab^{n+1}a, \text{ para todo } n \geq 1.$$

Por tanto, para el esquema H $\sigma = (V, R)$ y L, tenemos:

$$\sigma(L) = \{aba, abbba\},$$
$$\sigma^0(L) = \{abba\},$$
$$\sigma^1(L) = \{aba, abba, abbba\}, \text{ y}$$
$$\sigma^i(L) = \{ab^n a \mid 1 \leq n \leq i + 2\}, \text{ para } i \geq 1.$$

Por consiguiente:

$$\sigma^*(L) = \{ab^n a \mid n \geq 1\}.$$

Dadas dos familias de lenguajes FL_1, FL_2, definimos:

$$H(FL_1, FL_2) = \{\sigma^*(L) \mid L \in FL_1 \text{ y } \sigma = (V, R), \text{ con } R \in FL_2\}.$$

Los resultados esenciales en esta área, que es de crucial importancia para la computación con ADN basada en el empalme, son los dos siguientes:

Lema 1 (Lema de Preservación de la Regularidad) $H(REG, FIN) \subseteq REG$.

Así pues, con un conjunto finito de reglas de empalme no obtenemos nada más que lenguajes regulares. Dar el paso adelante más pequeño (en la jerarquía de Chomsky), esto es, emplear simplemente un conjunto regular de reglas de empalme, nos lleva a dar el máximo salto posible en capacidad generativa: conseguimos caracterizar los lenguajes recursivamente enumerables, módulo una operación de filtrado en forma de una intersección con un lenguaje regular.

Lema 2 (Lema Básico de Universalidad) *Todo lenguaje* $L \in RE, L \subseteq T^*$, *se puede escribir como* $L = L' \cap T^*$, *para algún* $L' \in H(FIN, REG)$.

El Lema 1 fue demostrado por primera vez en [4]; una prueba directa con autómatas finitos se ofrece en [36] (cfr. también [34]). El Lema 2 fue demostrado en [31]. Un resultado mucho más general que el del Lema 1 se da en [37]: toda AFL completa es cerrada por el empalme iterado con respecto a un esquema H finito.

La intersección con T^* en el Lema Básico de Universalidad sugiere la definición de un *sistema H extendido*, que es una cuadrupleta:

$$\gamma = (V, T, A, R),$$

donde V es un alfabeto, $T \subseteq V$, $A \subseteq V^*$ y $R \subseteq V^*\#V^*\$V^*\#V^*$, siendo $\#, \$$ símbolos especiales que no pertenecen a V.

Llamamos a V el *alfabeto* de γ, T es el alfabeto *terminal*, A el conjunto de *axiomas* y R el conjunto de *reglas de empalme*.

El *lenguaje generado* por γ se define como:

$$L(\gamma) = \sigma^*(A) \cap T^*,$$

donde $\sigma = (V, R)$ es el esquema H subyacente de γ.

Dadas dos familias de lenguajes FL_1, FL_2, denotamos mediante:

$$EH(FL_1, FL_2)$$

la familia de lenguajes $L(\gamma)$ generados por sistemas H extendidos $\gamma = (V, T, A, R)$, con $A \in FL_1, R \in FL_2$.

Es fácil demostrar el siguiente resultado complementario del Lema de Preservación de la Regularidad.

Lema 3 $REG \subseteq EH(FIN, FIN)$.

Los resultados conocidos sobre la capacidad generativa de los sistemas H extendidos se resumen en la Tabla 1 de la página 45, donde en la intersección de la fila FL_1 con la columna FL_2 aparece o bien la familia $EH(FL_1, FL_2)$ o dos familias FL_3, FL_4 tales que $FL_3 \subset EH(FL_1, FL_2) \subseteq FL_4$. Estas familias

	FIN	REG, LIN, CF, CS, RE
FIN	REG	RE
REG	REG	RE
LIN	LIN, CF	RE
CF	CF	RE
CS	RE	RE
RE	RE	RE

Tabla 1: La capacidad generativa de los sistemas H extendidos.

FL_3, FL_4 son las mejores estimaciones posibles de entre las seis familias que consideramos aquí.

Tal como se ve, la única familia que no coincide con una sola de las de la jerarquía de Chomsky es $EH(LIN, FIN)$. Obsérvese también que la familia de lenguajes independientes del contexto no tiene caracterización en términos de sistemas H. Recientemente [28], se ha obtenido una caracterización de ella en términos de sistemas H con reglas $u_1\#u_2\$u_3\#u_4$ cuyas cadenas u_1, u_2, u_3, u_4 son todas de longitud como máximo uno pero cuya aplicación está regulada por contextos permisivos (cada regla tiene ciertos símbolos asociados y se puede emplear para empalmar dos cadenas sólo si tales símbolos asociados están presentes en las mismas).

Estos sistemas H controlados, que son el equivalente de las gramáticas independientes del contexto reguladas en la teoría "clásica"de lenguajes formales, fueron propuestos para responder a la siguiente dificultad. De acuerdo con el Lema de Regularidad, los sistemas H finitos pueden computar a lo sumo al nivel de los autómatas finitos. Trabajar con conjuntos infinitos de reglas de empalme, como en el Lema Básico de Universalidad, no tiene interés práctico. En la demostración del Lema Básico de Universalidad se puede comprobar que el conjunto de reglas es infinito precisamente para controlar la aparición de ciertos símbolos en las cadenas empalmadas, y esto es exactamente para lo que sirven los contextos permisivos. Así pues, los sistemas H con conjuntos finitos de reglas de empalme controladas por contextos permisivos permiten caracterizar los lenguajes recursivamente enumerables. Se alcanza idéntico resultado si se usan contextos prohibitivos (estos dos tipos de sistemas H son los equivalentes a las gramáticas de contexto aleatorio). Se pueden encontrar muchas otras clases de sistemas H controlados en [34].

Un procedimiento alternativo para obtener sistemas H finitos que sean capaces de caracterizar los lenguajes recursivamente enumerables consiste en seguir el modelo que es habitual en la teoría de sistemas de gramáticas, a saber, distribuir los elementos del sistema H (los axiomas y las reglas de empalme) de tal manera que tengamos diversos sistemas H como componentes que cooperen de una manera determinada en la generación de un lenguaje único.

Nos remitimos de nuevo a [34] para los detalles, y solamente mencionaremos dos tipos de resultados que son comunes en este terreno: la caracterización de los lenguajes recursivamente enumerables mediante sistemas con un número pequeño de componentes y la misma caracterización por medio de sistemas con componentes pequeños (es decir, con pocas reglas en cada componente).

Todas estas caracterizaciones de los lenguajes recursivamente enumerables pueden ser consideradas como pruebas teóricas de la posibilidad de diseñar "ordenadores"de ADN basados en la operación de empalme que sean *programables*: las correspondientes clases de sistemas H exhiben ciertas propiedades de universalidad, esto es, existen mecanismos fijos que pueden simular cualquier otro mecanismo dado con sólo introducir un "código"del mecanismo simulado (como axioma) en aquél.

Otros Vínculos entre la Teoría de Lenguajes Formales y la Computación con ADN

Los dispositivos de generación de lenguajes que hemos considerado en las secciones anteriores se basan en operaciones con cadenas y lenguajes (anillamiento y empalme) que son nuevas en la teoría de lenguajes formales. De esta forma han nacido nuevas ramas de la teoría, la más desarrollada de las cuales es probablemente la teoría de los sistemas H, sobre los cuales se han publicado más de doscientos trabajos hasta el momento. Por otro lado, los lenguajes de intercalación gemela y los autómatas finitos de muchas cabezas son "material viejo"de la teoría de lenguajes formales. Asimismo, otras nociones antiguas en este mismo sentido resultan tener interpretaciones o aplicaciones interesantes en la computación molecular y ofrecen, de esta manera, motivaciones para nuevos desarrollos.

Un caso típico de ello son las operaciones de *inserción-supresión*. Estas operaciones fueron investigadas por razones lingüísticas ya en [14]. La inserción es también la operación esencial de las gramáticas contextuales de Marcus [22]. En el área del ADN, la inserción de pequeñas secuencias de nucleótidos es una operación frecuente *in vivo*, y también se puede ejecutar *in vitro* mediante el anillamiento no emparejante u otras técnicas. Los detalles se pueden encontrar en [20] y [19]; para un estudio amplio de los sistemas de inserción-supresión, remitimos a los capítulos correspondientes de [32] y [34].

Un caso parecido es el de la operación de *duplicación*. Pasar de xuy a $xuuy$ es tanto una operación simple en un lenguaje formal como una transformación posible de una secuencia de ADN. Lo que no es tan simple es el estudio del comportamiento de la duplicación con respecto a las clases de lenguajes de la jerarquía de Chomsky. Por ejemplo, iteremos esta operación. Partimos de un conjunto de cadenas, $L \subseteq V^*$, producimos todas las cadenas posibles obtenidas duplicando subcadenas de las cadenas de L e iteramos este procedimiento. Denotamos mediante $D(L)$ el lenguaje resultante. Cuando V contiene sola-

mente dos símbolos, si L es un lenguaje regular entonces $D(L)$ también lo es ([7]), pero para alfabetos arbitrarios esto no es necesariamente así ([46]): hay cadenas w tales que $D(\{w\})$ no es un lenguaje regular. Todavía no conocemos si $D(\{w\})$ es un lenguaje independiente del contexto para cualquier cadena w.

Otra relación interesante entre el ADN y la teoría de lenguajes formales, que también conduce a un resultado teórico muy sorprendente, la propone Salomaa en [44]: los *sistemas* D0L *de Watson-Crick*. La idea es bien simple. Al igual que en el caso del ADN, para cada alfabeto V consideremos un alfabeto gemelo, $\overline{V}$, que consta de las versiones con barra de los símbolos de V. Tomemos una cadena $w \in (V \cup \overline{V})^*$ y el morfismo g definido por $g(a) = \overline{a}, g(\overline{a}) = a$, para todo $a \in V$. Digamos que la cadena w es *mala* si contiene más símbolos con barra que sin barra. Si w es mala, obviamente $g(w)$ es *buena*. Un sistema D0L de Watson-Crick es un sistema D0L ordinario $G = (V, h, w)$, donde w es una cadena buena, que funciona como sigue: comienza con w y produce $h(w)$; si esta cadena es buena, entonces continúa; si es mala, la sustituye por $g(h(w))$ y continúa; y así sucesivamente. La única diferencia, por tanto, con respecto al modo usual de funcionamiento de un sistema D0L es que siempre que se obtiene una cadena mala la reemplazamos por su cadena complementaria y continuamos la reescritura a partir de ésta.

Es sabido que los sistemas D0L tienen una capacidad generativa bastante reducida (existen incluso lenguajes finitos que no son D0L, la función de crecimiento de un sistema D0L es de una forma bien determinada, etc.). En [44] se demuestra que el rasgo adicional de trabajar siempre con cadenas buenas, utilizando eventualmente el morfismo complementador g, incrementa la capacidad de los sistemas D0L. El nivel de incremento de capacidad no se ha conocido hasta muy recientemente, y el resultado es totalmente inesperado: todas las funciones recursivas pueden ser computadas por medio de sistemas D0L de Watson-Crick, en el sentido de que el paso de una cadena que codifica el argumento de la función a otra que codifica el valor de la misma para ese argumento puede ser ejecutado por un sistema D0L de Watson-Crick. La prueba de este resultado propio de la teoría de lenguajes formales está basada en la teoría de funciones recursivas y puede consultarse en [45]. Nos sirve para insistir de nuevo en el poder de la complementariedad, ya visible en los sistemas adherentes y en los autómatas de Watson-Crick, y es uno de los resultados más "puramente" de teoría de lenguajes formales sugeridos por la computación con ADN.

Sistemas P y Gramáticas Matriciales

Otro impulso para nuevos desarrollos, en una dirección no tradicional, procede del terreno de la computación con membranas. La idea original tiene que ver con la evidencia de que una célula viva puede ser vista como un mecanismo computacional, sin más que interpretar que los procesos químicos que tienen lugar en su compleja estructura son procesos de manipulación de

información. Ello ha llevado a considerar un modelo de computación (llamado *sistema P*) que consta de una *estructura de membranas* con *multiconjuntos* de objetos situados en las regiones de que se compone y que evolucionan conforme a ciertas *reglas de evolución*. Para los detalles, remitimos a [33], donde se introdujo el modelo, y a [3], que es un primer panorama general.

Se conocen muchas variedades de sistemas P, la mayoría de las cuales son computacionalmente universales (caracterizan la familia de conjuntos recursivamente enumerables de números naturales). La herramienta esencial en todas las demostraciones de universalidad es una forma normal clásica para gramáticas matriciales (propuesta ya en 1965).

Una *gramática matricial con comprobación de aparición* es una estructura $G = (N, T, S, M, F)$, donde N, T son alfabetos disjuntos, $S \in N$, M es un conjunto finito de secuencias de la forma $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, de reglas independientes del contexto sobre $N \cup T$ (con $A_i \in N, x_i \in (N \cup T)^*$, en todos los casos) y F es un conjunto de ocurrencias de reglas en M (N es el alfabeto *no terminal*, T es el alfabeto *terminal*, S es el *axioma*, y los elementos de M se llaman *matrices*).

Dados $w, z \in (N \cup T)^*$, escribimos $w \Longrightarrow z$ si existen una matriz $(A_1 \to x_1, \ldots, A_n \to x_n)$ de M y cadenas $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, tales que $w = w_1, z = w_{n+1}$ y, para cualesquiera i, $1 \leq i \leq n$, o bien $w_i = w'_i A_i w''_i, w_{i+1} = w'_i x_i w''_i$, con $w'_i, w''_i \in (N \cup T)^*$, o bien $w_i = w_{i+1}$, A_i no aparece en w_i y la regla $A_i \to x_i$ forma parte de F. (Las reglas de una matriz se aplican en orden, dejando de lado eventualmente las de F que no se puedan aplicar; decimos que se aplican en el modo de *comprobación de aparición*.)

El lenguaje generado por G se define como $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. La familia de lenguajes de esta forma se representa con MAT^{λ}_{ac}. Si $F = \emptyset$, se dice que G es sin comprobación de aparición. La familia de lenguajes generados por este último tipo de gramáticas se representa con MAT^{λ}. Si no se permiten reglas de borrado, quitamos el superíndice λ.

Es conocido que $CF \subset MAT \subset MAT_{ac} \subset CS \subset MAT^{\lambda}_{ac} = RE, MAT \subseteq MAT^{\lambda} \subset RE$. Otros detalles sobre las gramáticas matriciales se pueden encontrar en [8] y [40].

He aquí un ejemplo simple de una gramática matricial que genera el lenguaje no independiente del contexto $\{a^n b^n c^n \mid n \geq 1\}$, ya considerado en la sección Prerrequisitos de Teoría de Lenguajes Formales:

$$G = \left\{ \begin{array}{l} \{S, A, B, C\}, \{a, b, c\}, S, \emptyset, \\ \{(S \to ABC), (A \to aA, B \to bB, C \to cC),\ (A \to a, B \to b, C \to c)\} \end{array} \right\}$$

Obsérvese que esta gramática contiene una sola regla no regular, $S \to ABC$, y todas las demás son regulares: ello es ilustrativo de la potencia de las gramáticas matriciales, porque el lenguaje $L(G)$ es no independiente del contexto.

Se dice que una gramática matricial $G = (N, T, S, M, F)$ está en *forma normal binaria* si $N = N_1 \cup N_2 \cup \{S, \#\}$, siendo mutuamente disjuntos estos tres conjuntos, y las matrices de M son de alguna de las siguientes formas:

1. $(S \to XA)$, con $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, con $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$,
3. $(X \to Y, A \to \#)$, con $X, Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, con $X \in N_1, A \in N_2, x \in T^*$.

Además, hay que resaltar que existe una única matriz de tipo 1 y que F consta exactamente de todas las reglas $A \to \#$ que aparecen en las matrices de tipo 3. # es un símbolo-trampa: una vez introducido, ya no es posible quitarlo. Las matrices de tipo 4 se emplean sólo una vez, en el último paso de una derivación.

Según el Lema 1.3.7 de [8], para cada gramática matricial G existe una gramática matricial equivalente G' en forma normal binaria. Si G es sin comprobación de aparición, también lo es G'.

Sea $G = (N, T, S, M, F)$ una gramática matricial arbitraria y denotemos con $ac(G)$ la cardinalidad del conjunto $\{A \in N \mid A \to \alpha \in F\}$. A partir de la construcción de la prueba del Lema 1.3.7 de [8], se puede ver que si partimos de una gramática matricial G y obtenemos G' en forma normal binaria, entonces $ac(G') = ac(G)$.

Por [30] sabemos que cualquier lenguaje recursivamente enumerable puede ser generado por una gramática matricial con comprobación de aparición que contiene como máximo seis símbolos no terminales. Ahora bien, en la demostración de [30] todos estos símbolos no terminales se utilizan en el modo *ac*. Con ocasión de la aplicación en la computación con membranas, [12] ha mejorado el resultado de [30]: todo lenguaje recursivamente enumerable puede ser generado por una gramática matricial con un máximo de cinco símbolos no terminales de los cuales un máximo de cuatro se emplean en el modo *ac*. Útil para los sistemas P es otro resultado de [12]: si no nos fijamos en el número total de símbolos no terminales, sino en los que efectivamente se usan en las reglas de comprobación de aparición, vemos que son sólo tres.

En consecuencia, a las propiedades de una gramática matricial G en forma normal binaria podemos añadir el hecho de que $ac(G) \leq 3$. Esta es la llamada *forma normal binaria fuerte* para las gramáticas matriciales. Gracias a esta forma mejorada del resultado de [8], en [11] se demuestra que tres jerarquías sobre el número de membranas, de las que no se conocía antes si eran finitas o infinitas, realmente se colapsan, dado que en los tres casos bastan cinco membranas para obtener sistemas P universales. Como no tenemos la certeza de que este resultado sea óptimo, se nos plantea el siguiente doble problema abierto: cómo mejorar el resultado de [12] y/o cómo mejorar las pruebas de [11]. La interacción entre lo clásico y lo moderno se aprecia muy bien en este contexto.

Multiconjuntos que Reescriben

Muy relevante desde el punto de vista de este artículo es el hecho de que los sistemas P son una especie de sistemas de gramáticas, que se ocupan de generar un conjunto de multiconjuntos (en [21], a un conjunto así se le denomina *macroconjunto*) de una manera distribuida. Ello plantea el reto de procesar macroconjuntos de una manera no distribuida, en un estilo "a lo Chomsky", o, lo que es lo mismo, de reconstruir la teoría de lenguajes formales sin atención al orden de los símbolos e identificando aquellas cadenas que compartan la misma aplicación de Parikh.

Para comodidad del lector, traemos aquí algunas definiciones y resultados de [21] que dan una primera respuesta a la cuestión anterior.

Un multiconjunto sobre un alfabeto V es una aplicación $\mu : V \longrightarrow \mathbf{N}$. Representamos con $V^{\#}$ el conjunto de todos los multiconjuntos sobre V. La unión, intersección, inclusión y diferencia de dos multiconjuntos se definen de manera natural (cfr., e.g., [48]). Dado un multiconjunto μ, se denota mediante $|\mu|$ su *peso*, definido como $|\mu| = \sum_{a \in V} \mu(a)$; para $U \subseteq V$, escribimos $|\mu|_U = \sum_{a \in U} \mu(a)$.

Una *gramática de multiconjuntos* es una estructura $G = (N, T, A, P)$, donde N, T son alfabetos disjuntos, el *no terminal* y el *terminal* respectivamente (representamos su unión con V), $A \subseteq V^{\#}$ es un macroconjunto finito sobre V (sus elementos se denominan *axiomas*) y P es un conjunto finito de *reglas de reescritura de multiconjuntos* (abreviadamente, *reglas*) de la forma $\mu_1 \rightarrow \mu_2$, donde μ_1, μ_2 son multiconjuntos sobre V y $|\mu_1|_N \geq 1$.

Dados dos multiconjuntos α_1, α_2 sobre V, escribimos $\alpha_1 \Longrightarrow_r \alpha_2$, para alguna $r : \mu_1 \rightarrow \mu_2 \in P$, si $\mu_1 \subseteq \alpha_1$ y $\alpha_2 = (\alpha_1 - \mu_1) \cup \mu_2$. Si r se sobreentiende por el contexto, entonces escribimos $\Longrightarrow$ en lugar de $\Longrightarrow_r$. Se designa mediante $\Longrightarrow^*$ la clausura reflexiva y transitiva de la relación $\Longrightarrow$. El *macroconjunto generado* por G se define como:

$$M(G) = \{\beta \in T^{\#} \mid \alpha \Longrightarrow^* \beta, \text{ con } \alpha \in A\}.$$

Una clasificación natural "a lo Chomsky"de tales gramáticas es la siguiente:

1. Las gramáticas G como las de arriba se llaman *arbitrarias*.
2. Si $|\mu_1| \leq |\mu_2|$ para todas las reglas $\mu_1 \rightarrow \mu_2$ de P, entonces G recibe el nombre de *monótona*.
3. Si $|\mu_1| = 1$ para todas las reglas $\mu_1 \rightarrow \mu_2$ de P, entonces G se denomina *independiente del contexto*.
4. Si $|\mu_1| = 1$ y $|\mu_2|_N \leq 1$ para todas las reglas $\mu_1 \rightarrow \mu_2$ de P, entonces G es *lineal*.

No disponemos de un equivalente directo de las gramáticas regulares, pero podemos considerar la siguiente subclase de las gramáticas lineales:

5. Si G es una gramática lineal de multiconjuntos tal que, para cada regla $\mu_1 \to \mu_2$ de P, si $\mu_1(a_i) > 0$, $\mu_2(a_j) > 0$, con $1 \leq i < j \leq n$, entonces $\mu_2(a_k) > 0$ para cualquier $i \leq k \leq j$, en esas circunstancias G se llama *local*. (Siempre incrementamos el número de copias de los objetos que son adyacentes en la ordenación de V).

Designamos mediante *mARB, mMON, mCF, mLIN, mLOC* las familias de macroconjuntos generados por gramáticas de multiconjuntos arbitrarias, monótonas, independientes del contexto, lineales y locales, respectivamente. Dada una familia F de lenguajes, PsF denota la familia de conjuntos de Parikh de vectores asociados con lenguajes de F. La familia de todos los lenguajes semilineales se representa por $SLin$.

También podemos considerar *gramáticas matriciales de multiconjuntos*, como una extensión directa de la definición para cadenas. Siempre tomamos únicamente gramáticas matriciales con reglas independientes del contexto. Añadimos la letra m delante de las notaciones para las familias matriciales de cadenas, obteniendo así las notaciones correspondientes para las familias matriciales de macroconjuntos: $mMAT, mMAT^{\lambda}, mMAT_{ac}$, y $mMAT^{\lambda}_{ac}$.

Diversas relaciones entre estas familias fueron establecidas en [21], en la forma del siguiente:

Teorema 10 *(Teorema de Jerarquía de Macroconjuntos) Entre las familias de macroconjuntos se cumplen las relaciones expresadas en el diagrama de la Figura 10, donde las flechas indican inclusiones estrictas de las familias de abajo en las familias de arriba, con la única excepción de la flecha junto a la que hay un signo de interrogación, cuyo carácter de inclusión propia es todavía una cuestión abierta.*

Obsérvense en esta figura las igualdades $mMON = mMAT$, $mARB = mMAT^{\lambda}$, que contrastan con las inclusiones propias $MAT \subset CS$, $MAT^{\lambda} \subset RE$ para el caso de cadenas. Asimismo, es digno de mención que la inclusión estricta $mMON \subset mARB$ se sigue de un "viejo" resultado, probado en un contexto teórico completamente diferente [17].

Muchas preguntas permanecen abiertas en esta área. Creemos que el asunto merece una investigación más profunda, que conduzca posiblemente a una significativa FMT (teoría formal de macroconjuntos), siguiendo el modelo de la FLT (teoría de lenguajes formales).

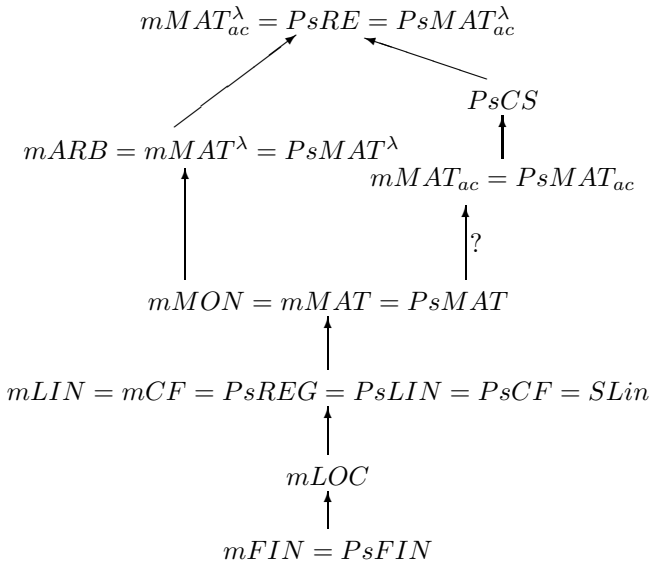


Figura 10. Jerarquía de familias de macroconjuntos.

Observaciones Finales

Hemos presentado varios temas que están en la zona de confluencia entre la teoría de lenguajes formales y la computación molecular (especialmente, la computación con ADN y la computación con membranas), como son: el "isomorfismo"entre las moléculas de ADN y los lenguajes de intercalación gemela, los sistemas adherentes y de empalme, los autómatas de Watson-Crick, los sistemas de inserción-supresión, el modelo YAC, los sistemas D0L de Watson-Crick y los sistemas P. Podríamos encontrar en la bibliografía bastantes otros temas de similares características. En ocasiones, las nociones de la teoría de lenguajes formales y de autómatas son interpretadas y usadas en el contexto de la computación con ADN o con membranas; otras veces, a la inversa, la manipulación del ADN *in vivo* o *in vitro* o la modelización del funcionamiento celular sugieren nuevos desarrollos en la teoría de lenguajes formales. Lo primero demuestra que la teoría de lenguajes formales almacena aún una cantidad importante de recursos no utilizados. Lo segundo permite prever un vigoroso desarrollo de la misma, con motivaciones nuevas vinculadas a la computación molecular, en un futuro cercano.

Referencias

[1] L. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, 226 (1994): 1021–1024.

[2] B. Baker y R. Book, Reversal-bounded multipushdown machines, *Journal of Computer and System Sciences*, 8 (1974): 315–332.

[3] C. Calude y Gh. Păun, *Computing with Cells and Atoms*. Taylor and Francis, London, 2000.

[4] K. Culik II, A purely homomorphic characterization of recursively enumerable sets, *Journal of the Association for Computing Machinery*, 26 (1979): 345–350.

[5] K. Culik II y T. Harju, Splicing semigroups of dominoes and DNA, *Discrete Applied Mathematics*, 31 (1991): 261–277.

[6] J. Dassow y V. Mitrana, On some operations suggested by the genome evolution, in R. Altmann et al., eds., *Proceedings of the Pacific Symposium on Biocomputing*, Hawai, Estados Unidos: 97–108. World Scientific, Singapore, 1997.

[7] J. Dassow, V. Mitrana y Gh. Păun, On the regularity of duplication closure, *Bulletin of the European Association for Theoretical Computer Science*, 68 (1999): 133–136.

[8] J. Dassow y Gh. Păun, *Regulated Rewriting in Formal Language Theory*. Springer, Berlin, 1989.

[9] J. Engelfriet, Reverse twin-shuffles, *Bulletin of the European Association for Theoretical Computer Science*, 60 (1996): 144.

[10] J. Engelfriet y G. Rozenberg, Fixed point languages, equality languages, and representations of recursively enumerable languages, *Journal of the Association for Computing Machinery*, 27 (1980): 499–518.

[11] R. Freund, C. Martín-Vide y Gh. Păun, From regulated rewriting to computing with membranes: collapsing hierarchies, ms. inédito, 2001.

[12] R. Freund y Gh. Păun, On the number of variables in graph-grammars, programmed grammars, and matrix grammars, en *Proceedings of the Conference on Universal Machines and Computations*, Chişinău, Lecture Notes in Computer Science 2055, Springer, Berlin, 2001.

[13] R. Freund, Gh. Păun, G. Rozenberg y A. Salomaa, Watson-Crick finite automata, en [47]: 305–317.

[14] B.S. Galiukschov, Semicontextual grammars (en ruso), *Mat. logica i mat. ling.*, 1981: 38–50.

[15] V. Geffert, Normal forms for phrase-structure grammars, *RAIRO. Theoretical Informatics and Applications*, 25 (1991): 473–496.

[16] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bulletin of Mathematical Biology*, 49 (1987): 737–759.

[17] N.D. Jones, L.H. Landweber y Y.E. Lien, Complexity of some problems in Petri nets, *Theoretical Computer Science*, 4 (1977): 277–299.

[18] L. Kari, Gh. Păun, G. Rozenberg, A. Salomaa y S. Yu, DNA computing, sticker systems, and universality, *Acta Informatica*, 35 (1998): 401–420.

[19] L. Kari, Gh. Păun, G. Thierrin y S. Yu, At the crossroads of DNA computing and formal languages: characterizing RE using insertion-deletion systems, en [47]: 318–333.

[20] L. Kari y G. Thierrin, Contextual insertion/deletion and computability, *Information and Computation*, 131.1 (1996): 47–61.

[21] M. Kudlek, C. Martín-Vide y Gh. Păun, Toward FMT (Formal Macroset Theory), en *Pre-Proceedings of the Workshop on Multiset Processing*, Curtea de Argeş, Rumanía, Technical Report CDMTCS-140: 149–158. University of Auckland, Auckland.

[22] S. Marcus, Contextual grammars, *Revue Roumaine des Mathématiques Pures et Appliquées*, 14 (1969): 1525–1534.

[23] S. Marcus, Linguistic structures and generative devices in molecular genetics, *Cahiers de Linguistique Théorique et Appliquée*, 11 (1974): 77–104.

[24] S. Marcus, Language at the crossroad of computation and biology, en Gh. Păun, ed., *Computing with Bio-Molecules: Theory and Experiments*: 1–35. Springer, Berlin, 1998.

[25] C. Martín-Vide, Natural computation for natural language, *Fundamenta Informaticae*, 31.2 (1997): 117–124.

[26] C. Martín-Vide y Gh. Păun, Normal forms for Watson-Crick finite automata, en F. Cavoto, ed., *The Complete Linguist: A Collection of Papers in Honour of Alexis Manaster Ramer*. Lincom Europa, Munich, 2001.

[27] A. Mateescu, Gh. Păun y A. Salomaa, Variants of twin-shuffle languages, ms. inédito, 1998.

[28] T. Muralidhar y K. Krithivasan, The generative power of simple H systems with permitting contexts, *Theoretical Computer Science*, en prensa.

[29] A. Păun y M. Păun, State and transition complexity of Watson-Crick finite automata, en G. Ciobanu, Gh. Păun, eds., *Fundamentals of Computation Theory, FCT'99*, Iaşi, Rumanía, Lecture Notes in Computer Science 1684: 409–420. Springer, Berlin, 1999.

[30] Gh. Păun, Six nonterminals are enough for generating each r.e. language by a matrix grammar, *International Journal of Computer Mathematics*, 15 (1984): 23–37.

[31] Gh. Păun, Regular extended H systems are computationally universal, *Journal of Automata, Languages, and Combinatorics*, 1 (1996): 27–36.

[32] Gh. Păun, *Marcus Contextual Grammars*. Kluwer, Dordrecht, 1997.

[33] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61.1 (2000): 108–143. (También TUCS Research Reports, 208, noviembre 1998, http://www.tucs.fi)

[34] Gh. Păun, G. Rozenberg y A. Salomaa, *DNA Computing: New Computing Paradigms*. Springer, Berlin, 1998.

[35] Gh. Păun, G. Rozenberg y T. Yokomori, Hairpin Languages, *International Journal of Foundations of Computer Science*, en prensa.

[36] D. Pixton, Regularity of splicing languages, *Discrete Applied Mathematics*, 69 (1996): 101–124.

[37] D. Pixton, Splicing in abstract families of languages, *Theoretical Computer Science*, 234.1-2 (2000): 135–166.

[38] A.L. Rosenberg, On multihead finite automata, *IBM Journal of Research and Development*, 10 (1966): 388–394.

[39] G. Rozenberg y A. Salomaa, Watson-Crick complementarity, universal computations and genetic engineering, *Technical Reports*, 96-28, Department of Computer Science, Leiden University, 1996.

[40] G. Rozenberg y A. Salomaa, eds., *Handbook of Formal Languages*, 3 vols. Springer, Berlin, 1997.

[41] K. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori y M. Hagiya, Molecular computation by DNA hairpin formation, *Science*, 288 (19 mayo 2000): 1223–1226.

[42] A. Salomaa, Equality sets for homomorphisms of free monoids, *Acta Cybernetica*, 4 (1978): 127–139.

[43] A. Salomaa, *Jewels of Formal Language Theory*. Computer Science Press, Rockville, MD, 1981.

[44] A. Salomaa, Turing, Watson-Crick, and Lindenmayer: aspects of DNA complementarity, en C.S. Calude, J. Casti, M.J. Dinneen, eds., *Unconventional Models of Computation*: 94–107. Springer, Singapore, 1998.

[45] P. Sosik, Universal computation with Watson-Crick D0L systems, ms. inédito, 2000.

[46] M. Wang, On the irregularity of the duplication closure, *Bulletin of the European Association for Theoretical Science*, 70 (2000): 162–163.

[47] D. Wood, L. Kari, R. Lipton, J. Reif, N. Seeman y E. Winfree, eds., *Preliminary Proceedings of the 3rd DIMACS Workshop on DNA Based Computers*. University of Pennsylvania, Philadelphia, 1997.

[48] R.R. Yager, On the theory of bags, *International Journal of General Systems*, *13* (1986): 23–37.

[49] T. Yokomori, YAC: Yet another computation model of self-assembly, en E. Winfree, D. Gifford, eds., *Preliminary Proceedings of the 5th International Workshop on DNA Based Computers*: 153–168. MIT Press, Cambridge, MA, 1999.

Carlos Martín–Vide
Grupo de Investigación en Lingüística Matemática
Universidad Rovira i Virgili
Pl. Imperial Tàrraco, 1
43005 Tarragona
correo electrónico: cmv@astor.urv.es

Gheorghe Păun
Romanian Academy of Sciences
PO Box 1-764
70700 Bucarest, Rumanía
correo electrónico: gpaun@imar.ro
g_paun@hotmail.com