Azar y Autómatas
Algunas demostraciones

October 28, 2019

1 Cardinalidad

Veamos la relación de cardinalidad de un conjunto y su conjunto potencia.

Theorem 1. *Sea X cualquier conjunto.* $|X| < |\mathcal{P}(X)|$.

Proof. Claramente $|X| \leq |\mathcal{P}(X)|$, ya que para cada $x \in X$, $f(x) = \{x\}$ proporciona una función inyectiva. Debemos ver que no podemos definir ninguna función suryectiva de X en $\mathcal{P}(X)$. Supongamos que sí pudiermaos definirla, $f : X \to \mathcal{P}(X)$, suryectiva.

Sea $Y = \{x \in X : x \not\in f(x)\}$. Como $Y \subseteq X$ entonces $Y \in \mathcal{P}(X)$. Como supusimos que f es suryectiva entonces existe $a \in X$ tal que $f(a) = Y$. Hay dos casos posibles:

1. $a \in f(a)$. Pero es imposible ya que $Y = f(a)$, contradice la definición de Y.
2. $a \notin f(a)$, que por definición de Y, obtenemos $a \in Y$. Pero $Y = f(a)$, entonces $a \in f(a)$, contradicción.

Luego f no puede ser suryectiva sobre $\mathcal{P}(X)$, y concluímos $|X| < |\mathcal{P}(X)|$. □

Como corolario obtenemos que la cardinalidad del conjunto N de números naturales es menor o igual a la de cualquier conjunto infinito, y que el conjunto potencia de cualquier conjunto infinito es incontable.

2 H, la complejidad descriptiva

Observation 2. *Las secuencias computables poseen muy baja complejidad. Si S es computable S satisface*

$$\exists C \ \forall i \quad H(S_i) \leq H(i) + C$$

Observation 3. *Toda cadena S que satisface*

$$\exists C \ \forall i \quad H(S_i) \leq H(i) + C$$

tambien satisface

$$\exists D \ \forall i \quad H(S_i/i) \leq D$$

Proof. Sea S una secuencia infinita S que satisface

$$\exists C \ \forall i \quad H(S_i) \leq H(i) + C$$

Por Teorema *** se cumple siempre que :

$$H(u, v) = H(u/v) + H(v) + O(1) = H(v/u) + H(u) + O(1)$$

Aplicamos el teorema a $H(S_i, i)$., la complejidad del par formado por un prefijo de S y su longitud.

$$H(S_i, i) = H(i/S_i) + H(S_i) + O(1) = H(S_i/i) + H(i) + O(1)$$

Sabemos que la complejidad de calcular el largo de una cadena cualquiera dado un programa mínimo para la cadena es un valor constante (es la longitud del programa más corto que toma el programa mínimo para calcular la cadena, la computa, calcula su longitud, la retorna y termina). Es decir,

$$\exists k \forall w \quad H(|w|/w) \leq k$$

Luego,

$$H(i/S_i) \leq k$$

Por lo tanto,

$$H(S_i, i) = H(S_i) + O(1)$$

Por hipótesis $\exists C \ \forall i \quad H(S_i) \leq H(i) + C$, entonces

$$H(S_i, i) = H(S_i) + O(1) \leq H(i) + O(1)$$

Luego,

$$H(S_i/i) + H(i) + O(1) = H(S_i) + O(1) \leq H(i) + O(1)$$

Lo que implica,

$$H(S_i/i) = O(1)$$

□

3 Conjuntos libres de prefijos y Computadoras

Theorem 4 (Desigualdad de Kraft)**.** *Si L es un conjunto libre de prefijos entonces*

$$\sum_{w \in L} 2^{-|w|} \leq 1.$$

Proof. Esta demostración se debe a Gregory Chaitin, y se basa en la interpretación geomtrica de un lenguaje $L \subseteq \sum^*$.

Consideremos $[0, 1]$. La prueba consiste en mostrar que si asignamos a cada palabra de L un subintervalo de $[0, 1]$ de longitud $2^-|w|$ entonces, si L es libre de prefijos los intervalos asignados no se superponene entre sí y por lo tanto la suma de las longitudes de los subintervalos asignados es menor que 1.

Para cada longitud n dividimos el $[0, 1]$ en 2^{-n} subintervalos. Sabemos que en $0, 1^*$ hay exactamente 2^n palabras de longitud n; si consideramos las 2^n palabras de longitud n ordenadas lexicográficamente nos permite asociar de manera natural a cada palabra con un ordinal k. Por ejemplo para $w = 000$, $k = 0$ ya que 000 es la primera palabra de longitud 3 mientras que 100 es la quinta ($k = 4$). Ahora sí podemos indicar la forma de asignar a cada palabra $w \in L$ un subinter un subintervalo de $[0, 1]$, mediante la siguiente función:

$$f(w) = [k2^{-|w|}, (k + 1)2^{-|w|}]$$

considerando que w es la k-ésima palabra en el orden canónico restringido a las palabras de la misma longitud que w.

Observation 5. *La función f asigna intervalos que son disjuntos o se incluyen unos a otros. Formalmente, $\forall v, w \quad f(w) \subseteq f(v)$ o $f(w) \subseteq f(v)$ o $f(v) \cap f(w) = \emptyset$.*

Sean v, w dos palabras arbitrarias en $0, 1^*$ tales que sus intervalos se intersecan propiamente. Supongamos $|v| = m$ y $|w| = n$. $f(v) = [k2^{-m}, (k + 1)2^{-m}]$ y $f(w) = [l2^{-n}, (l + 1)2^{-n}]$. Como los intervalos tienen intersección no vacía tenemos

a) $k2^{-m} < l2^{-n} < (k + 1)2^{-m} < (l + 1)2^{-n}$

b) $l2^{-n} < k2^{-m} < (l + 1)2^{-n} < (k + 1)2^{-m}]$.

□

4 Ejemplos

4.1 Una secuencia finita aleatoria

Como sabemos la noción de máxima complejidad o incompresibilidad de cadenas finitas no es categórica, sino que las cadenas son incompresibles en cierto "grado". Por ejemplo podemos decir que las cadenas finitas son incompresibles si todas sus descripciones tienen una longitud de más del 90 por ciento de la longitud de la cadena misma. En general la noción de incompresibilidad sobre cadenas finitas requiere especificar un parámetro que determina el límite entre lo compresible y lo incompresible.

Ahora deremos un ejemplo de una cadena finita aleatoria, que es la cantidad de cadenas aleatorias sobre cierta longitud. Fijemos n como la longitud y fijemos C la constante con signo que determina el límite entre las cadenas compresibles y las que no. Es decir, para toda cadena s de longitud n, s es compresible si y solo si $H(s) < n + H(n) + C$. Notar que C es típicamente un número negativo. Sea M la cantidad de cadenas incompresibles de largo n. Vamos a demostrar que M también es incompresible.

Sabemos que hay exactamente 2^n cadenas de largo n. Luego $0 \leq M \leq 2^n$. Expresaremos M como una cedena de n bits, rellenando con ceros a la izquierda en caso de ser necesario.

Supongamos el siguiente programa P que recibe como tres datos de entrada: n^*, el programa mínimo para n, Δ, ;a diferencia entre $n + H(n)$ y la complejidad máxima de cadenas de n bits, y q, un programa mínimo para M dado $n*$.

1. Calcula $H(n)$.
2. Computa n.
3. Computa M.
4. Encontrar $2^n - M$ cadenas compresibles de largo n. Esto se realiza ejecutando todos los programas de longitud menor que $n + \Delta$ hasta encontrar $2^n - M$ de ellos que se detienen y quedarse con su output.
5. Imprime x la primer cadena incompresible de largo n, según el orden lexicográfico. Notar que si Δ es un número negativo entonces hay a lo sumo $2^{(n+\Delta)} - 1$ cadenas compresibles, y por lo tanto hay certeza de encontrar cadenas incompresibles.)
6. Termina.

Sabemos que la complejidad de la cadena x retornada por este programa es

$$n + H(n) + \Delta \leq H(x).$$

Por otro lado hemos computado x mediante este programa p que recibe en forma de datos n^*, Δ y q.

$$H(x) \leq H(x/n, \Delta, q) + H(n, \Delta, q) \leq |p| + H(n) + \Delta + H(M/n)$$

$$n + H(n) + \Delta \leq |p| + H(n) + \Delta + H(M/n)$$

$$n \leq |p| + H(M/n)$$

$$H(M/n) > n - |p|$$

Luego, M dado un programa mínimo para n requiere un programa de largo n menos la longitud del programa p. Notemos que la longitud del programa p no depende de n.

Observemos ahora que para valores grandes de n la longitud de p se hace despreciable, y por lo tanto para expresar M dado n^* hacen falta todos los n bits. Esto significa que M no puede tener demasiados ceros a la izquierda, ya que en este caso M sería compresible. Concluímos que una fracción fija (mayor a 2^{n-C}) de las cadenas de cada longitud son incompresibles.

****

Ver : On the number of n bit strings of maimal complexity
Greg Chaitin
****

5 Los programas elegantes son aleatorios

EJEMPLO. Los programas autodelimitantes elegantes son aleatorios.

Proof. Sea p un programa elegante. Entonces

$$U(p, \lambda) = s \text{ y } \forall p' \text{ si } U(p', \lambda) = s \text{ entonces } |p| \leq |p'|.$$

Sabemos que para toda cadena existe un programa elegante que la retorna, entonces en particular, existe $p*$ un programa elegante que retorna p. Sea $b_1 b_2 \dots b_k p^*$ un programa que lee $p*$, produce p, computa p y retorna s, es decir, $U(b_1 b_2 \dots b_k p*) = s$. Como p es elegante debe cumplirse que

$$|p| \leq |b_1 b_2 \dots b_k p^*| = k + |p^*| = k + H(p)$$

.

Concluímos que

$$H(p) \geq |p| - k$$

Notemos que la constante k es independiente de p y por lo tanto los programas elegantes son todos incompresibles módulo la misma constante k.

□

5.1 Una secuencia no computable no aleatoria

Asumamos una enumeración fija de todas las máquinas de Turing, $TM_1, TM_2, TM_3, \ldots$. Sea A el número característico de la detención, es decir, $A : a_1 a_2 a_3 \ldots$ donde $a_i = 1$ si y solo si la i-esima máquina de Turing con la cinta vaciía se detiene. Veamos que A no es un número aleatorio. Sea j la cantidad de máquinas de Turing que se detienen con la cinta vacía entre las primeras i máquinas. Por lo tanto, $j \leq i$.

$$\forall i H(A_i) \leq H(A_i/i, j) + H(i, j) + O(1)$$

$$H(i, j) \leq H(i) + H(j) + O(1)$$

$$H(i) \leq 2\ log\ i + O(1)$$

$$H(j) \leq 2\ log\ j + O(1) \leq 2\ log\ i + O(1)$$

Y sabemos que $H(A_i/i, j)$ es una constante, es la longitud del pregrama más corto que realiza "dovetailing" entre las primeras i máquinas hasta encontrar las j que se detienen y así consigue cada uno de los bitas de A_i. Luego,

$$\forall i H(A_i) \leq 4\ log\ i + O(1)$$

Consecuentemente A no satisface que $\exists C \forall i\ H(A_i) > i - C.$, por lo que A no es una secuencia aleatoria.

5.2 Otra secuencia no computable no aleatoria

Siguiendo a Turing en su célebre artículo del 36 "On computable numbers with an application to the Entscheidungsproblem", los números reales computables pueden ser descriptos como los números cuya expansión decimal es calculable por medios finitos, es decir mediante un algoritmo. Digamos que un número real es computable si hay un programa que al ejecutarse va dando la lista de sus dígitos, uno por uno. Ejemplos de números computables son π, e y las raíces reales de polinomios de coeficientes enteros. Aunque la clase de los números computables es grande, es sin embargo, numerable. Como los programas son directamente secuencias de ceros y unos, números naturales en base dos, claramente hay tan solo una cantidad numerable de programas , y por lo tanto hay a lo sumo una cantidad numerable de números reales computables en el intervalo (0,1). Turing nos muestra que los números computables no incluyen todos los números definibles, y consigue definir un número que no es computable. Veamos cómo lo hace.

Definamos una lista de todos los posibles programas, y junto a cada programa ponemos o bien la sucesión (infinita) de dígitos del número real que computa, o bien un blanco en cualquier otra opción (si no computa un número real entre (0,1), o si se detiene en algún dígito, o entra en ciclo, o da cualquier señal de error). Tenemos una tabla del tipo:

p1	0,010010001...
p2:	
p3	0,333333...
p4:	1
p5 (π-3) :	0,141592...
.	
.	
.	

Es claro que todos los números reales computables en el intervalo (0,1) están en algunas de estas filas. Lo que hacemos a continuación es reproducir el argumento diagonal de Cantor. Es decir, definimos un nuevo número real entre 0 y 1, distinto a todos los listados, modificando los dígitos de la diagonal. Más precisamente llamemos $\beta = 0, b_1 b_2 b_3 ... b_n ...$ a este número. Cada b_i es distinto al i-ésimo dígito computado por el i-ésimo programa. Es decir, b_1 es distinto al primer dígito arrojado por el primer programa (en nuestro caso sería b_1 distinto de 1); b_2 es distinto al segundo dígito del segundo programa (en nuestro caso b_2 podría ser cualquier dígito porque $p2$ tiene su salida en blanco); etc. De esta manera hemos construido un número real entre 0 y 1 distinto a todos los listados en el diagrama. Por lo tanto β no es computable. Ahora bien, Por qué b no es computable, si prácticamente hemos indicado un método para obtener sus dígitos? Lo que ocurre es que para computar los dígitos de β necesitaríamos poder decidir si el siguiente programa en la lista arrojará o no un siguiente dígito. En otras palabras, Qué ocurre si al ejecutarse el programa i-ésimo no aparece su i-ésimo dígito? Esto podrá deberse a que el programa entró en un ciclo y nunca lo arrojará, o bien a que no se ha esperado el tiempo suficiente para que ese dígito sea procesado. Extraordinariamente, Turing se dio cuenta de que esta pregunta a priori no puede responderse. Si pudiera decidirse sobre esta cuestión obtendríamos el dígito i-ésimo de β de acuerdo con lo prescripto. Es decir, β sería computable! y tendría que estar en la tabla.

Este es el argumento original de Turing que le permite afirmar que, en general, no se puede decidir si un programa va a computar el dígito i-ésimo de un número real. Turing sabía que este mismo argumento valía en situaciones más generales y que se refería en el fondo a la imposibilidad de decidir si un programa cualquiera se detendrá o no para un valor dado de entrada. Es en el intento de precisar este argumento que define la noción de lo que se conoce ahora como máquina de Turing, a partir de la que luego demuestra la indecidibilidad del problema de la detención Hasta aquí tenemos los números reales divididos en computables y no computables, y el ejemplo de Turing de número no computable, β.

Ahora definamos β en forma unívoca, y veamos que no es un número aleatorio. Llamemos $t_{i,j}$ al dígito que se encuentra en la i-ésima fila j-ésima columna. Entonces $t_{i,j}$ será el j-ésimo dígito arrojado por el i-ésimo programa, o bien un 0 en caso de que el programa nunca vaya a arrojar

un j-ésimo dígito. Ahora definimos $\beta = b_1 b_2 b_3 \ldots$, donde $\forall i \;\; b_i = (t_{i,j} + 1) mod 10$.

Sea j la cantidad de programas entre los primeros i, que cuando son ejecutados con la cinta sin datos, imprimen suficientes dígitos hasta pisar la diagonal o bien se detienen antes de cruzarla. Por lo tanto, $j \leq i$, y tenemos que: dados i, j, haciendo "dovetailing" entre los primeros i programas se consiguen j valores de la diagonal, por lo que los demás valores hasta $t_{i,i}$ son 0s. Una vez obtenidos los valores t_{11} a t_{ii} resulta inmediato calcular $\beta_i = b_1 \ldots b_i$, donde $\forall i \;\; b_i = (t_{i,j} + 1) mod 10$.

$$\forall i H(\beta_i) \leq H(\beta_i / i, j) + H(i, j) + O(1)$$

$$H(i, j) \leq H(i) + H(j) + O(1)$$

$$H(i) \leq 2 \; log \; i + O(1)$$

$$H(j) \leq 2 \; log \; j + O(1) \leq 2 \; log \; i + O(1)$$

$$\forall i H(\beta_i) \leq 4 \; log \; i + O(1)$$

Consecuentemente β no satisface que $\exists C \forall i \;\; H(\beta_i) > i - C$, por lo que β no es una secuencia aleatoria.

5.3 Ω una secuencia aleatoria

Theorem 6. *El número* $\Omega = \sum_{U(p)\downarrow} 2^{-(|p|)}$ *es aleatorio.*

Sea p el siguiente programa: $b_1 b_2 \dots b_k \Omega_i^*$. Consideremos $g : N \to \sum^*$ una enumeración de los programas que se detienen con la cinta vacía.

Leer Ω_i* y computar Ω_i.
m=0
mientras $\sum_{j=1}^{m+1} 2^{-|g(j)|} \leq \Omega_i$
$m = m + 1$;

Computar $O = \{U(g(j)) : 1 \leq j \leq m\}$ el conjunto de los outputs de los primeros m programas que se detienen con la cinta vacía.
Sea n el mínimo número entero tal que $U(g(n)) \notin O$.

Veamos que $|g(n)| > i$. Para demostrarlo supongamos lo contrario, es decir, $|g(n)| \leq i$. Entonces,

$$\begin{aligned}
\Omega &> \sum_{j=1}^{m} 2^{-|g(n)|}, && \text{Porque } \Omega \text{ tiene infinitos aportes} \\
&> \Omega_i + 2^{-|g(n)|} && \text{Porque } \sum_{j=1}^{m} 2^{-|g(j)|} > \Omega_i \\
&\geq \Omega_i + 2^{-i} && \text{Porque supusimos } |g(n)| \leq i \\
&\geq \Omega && \text{Porque } \Omega_i \text{ contiene exactamente los primeros } i \text{ bits de } \Omega.
\end{aligned}$$

Llegamos a que $\Omega > \Omega$, y esto es imposible. Por lo tanto, $|g(n)| > i$.

Luego,

$$i \leq H(U(g(n)) \leq k + |\Omega_i^*| = k + H(\Omega_i)$$

Concluímos que para todo i,

$$H(\Omega_i) > i - k$$

5.4 Parecen aleatorios pero no lo son

Ejemplo.

$$\alpha = \sum_{U(p,\lambda) \text{ se detiene o } p \text{ es primo}} 2^{-|p|}$$

Este número α no es aleatorio porque existe una máquina universal para la cual puedo calcular cada prefijo de α.

Sea la siguiente función $g : N^* \to N^*$ que manda los primos en su ordinalidad y los no primos al 0. Es decir $g(1) = 1, g(2) = 2, g(3) = 3, g(4) = 0, \ldots g(7) = 5, g(8) = 0, \ldots$.

Sea U' la siguiente máquina Universal:

$\forall p, v$ si p es primo $U'(p, v) = U(g(p), v)$
$\forall p, v$ si p no es es primo $U'(p, v)$ no se detiene.

Ejemplo.

$$\gamma = \sum_{U(p,\lambda) \text{ se detiene y } p \text{ es primo}} 2^{-|p|}$$

Este número γ no es aleatorio porque existe una máquina universal para la cual puedo calcular cada prefijo de γ.

Sea la siguiente función $g : N^* \to N^*$ que manda los no primos en su ordinalidad y los primos al 0. Es decir $g(1) = 0, g(2) = 0, g(3) = 0, g(4) = 1, \ldots g(7) = 0, g(8) = 3, \ldots$.

Sea U' la siguiente máquina Universal:

$\forall p, v$ si p es primo $U'(p, v) = v$ (función identidad)
$\forall p, v$ si p no es primo $U'(p, v) = U(g(p), v)$.

$$\gamma = \sum_{U'(p,\lambda) \text{ se detiene y } p \text{ es primo}} 2^{-|p|} = \sum_{U(p,\lambda)\ p \text{ es primo}} 2^{-|p|}$$