Una segunda excursión a la entropía algorítmica de conjuntos

Mariano M. Moscato

22 de junio de 2005

Índice general

Introducción v

I Cómputos finitos 1

1. Máquinas de cómputo finito 3
- 1.1. Máquinas para cadenas . . . 3
- 1.2. Máquinas para conjuntos . . . 4
- 1.3. Máquinas universales . . . 5
- 1.4. Definiciones fundamentales . . . 6

2. Teorema de la codificación 11
- 2.1. Teorema de la codificación para palabras . . . 11
- 2.2. Teorema de la codificación para conjuntos . . . 14

3. Condiciones abstractas 17
- 3.1. Representaciones totales . . . 18
- 3.2. Representación parcial . . . 19

II Cómputos eternos 23

4. Máquinas de cómputos eternos 25
- 4.1. Máquinas sobre cadenas . . . 25
- 4.2. Máquinas sobre conjuntos . . . 26
- 4.3. Máquinas universales . . . 27
- 4.4. Definiciones fundamentales . . . 27

5. Teorema de la codificación 29
- 5.1. Nombres . . . 30
  - 5.1.1. Descubriendo diferencias . . . 33
- 5.2. Familias uniformes . . . 36
- 5.3. Familias consecutivas . . . 38
  - 5.3.1. Una cota superior para programas mínimos . . . 38
  - 5.3.2. Codificación en familias consecutivas . . . 45

5.3.3. Un intento fallido 54

6. Condiciones abstractas **57**
6.1. Destilando la estrategia 57
6.2. Aplicando la estrategia 59

Introducción

> No debo buscar mi dignidad en el espacio, sino en el gobierno de mi pensamiento. No tendré más aunque posea mundos. Si fuera por el espacio, el universo me rodearía y se me tragaría como a un átomo; pero por el pensamiento yo abrazo el mundo.
>
> Blas Pascal en *Pensées*

El razonamiento es un proceso que llama nuestra atención desde hace tiempo. La capacidad de procesar la información de la que disponemos gracias a nuestros sentidos, y que nos permitió diferenciarnos del resto del reino animal, ha despertado nuestra curiosidad.

En ese sentido, (algunos) han tratado de empezar por poco y sólo concentrarse en cierto tipo de pensamiento.

> Podemos comparar a un hombre [en el proceso de realizar una tarea rutinaria] con una máquina que sólo es capaz de un número finito de estados [...]
>
> A. M. Turing

Así se han definido diferentes artefactos imaginarios que parecerían disponer de (algunas de) las cualidades y mecanismos del pensamiento humano, al menos de las que utilizamos para llevar a cabo labores simples, como preparar la lista de las compras o calcular cuánto dinero necesitamos llevar al mercado (y ciertamente algunas un tanto más complejas, pero por ahora dejémoslas de lado).

Una vez identificado el objeto de estudio, las preguntas salieron impetuosas al encuentro de aquellos que se acercaron a él y, de acuerdo a desde qué lado se acercaba el explorador, distinta era la cara que se mostraba del problema y distinta la manera en que se materializaban las preguntas.

Información

La senda que guiará nuestro acercamiento puede descubrirse como la que nos pide ver a este artefacto, a esta máquina imaginaria que computa, como un procesador de información. Un dispositivo que toma un objeto y, aplicando

cierto proceso, lo transforma en otro. Estos artefactos se comportan como una función: por cada objeto que le brindemos nos devolverá sólo uno.

Por ejemplo, podríamos tener una máquina que dada una tira de números nos devuelva otra con los mismos números pero ordenados de mayor a menor. O bien, podríamos tener una máquina que dada una jugada de "ta-te-ti" nos diga si el juego terminó.

Podemos pensar que cada objeto (en general: una tira de letras, un conjunto de números, vos, yo, etc.) contiene cierta información. Pero, ¿cuánta? ¿De qué manera podemos medirla?

Una forma de hacerlo es aprovechar las computadoras de las que hablábamos para realizar estas mediciones. Así, la información de un objeto o tiene que ver con ciertas propiedades de aquellos objetos p que brindados a alguna máquina M provocan que su proceso resulte en o.

Podemos definir la probabilidad de obtener o en una máquina M (notada $P_M(o)$) como la probabilidad de que un proceso de cómputo de M resulte en o si se le brinda a la computadora un programa p escogido "al azar". Ligada a esta definición, hallamos la noción de entropía de un objeto o bajo la máquina M, definida como $H_M(o) = -\log P_M(o)$.

Paralelamente, podemos definir la longitud del programa mínimo para un objeto r en la máquina M, notada $I_M(r)$.

Un resultado en el que centraremos nuestra atención especialmente en el transcurso de este trabajo es aquel que une las nociones de programa mínimo para un objeto con su entropía, usualmente llamado *teorema de codificación*.

Los trabajos realizados tomando como objeto de estudio máquinas como las que mencionamos anteriormente, por lo general trabajan con un modelo en particular de estos aparatos. Si bien definiremos estos artefactos en detalle más adelante, nos interesará por ahora mencionar que los datos con los que trabajan son tiras de símbolos; en particular, palabras escritas con ceros y unos.

Chaitin [3] brinda una versión de este teorema para el caso de las cadenas de ceros y unos, cuya demostración repasaremos en el capítulo 2.

Antes de continuar, veamos un poco de notación que nos acompañará a lo largo de todo el trabajo. Dadas dos funciones f y g, notaremos $f(x) = g(x) + O(1)$ cuando exista una constante c tal que, para todo x, $g(x) - c \leq f(x) \leq g(x) + c$.

Teorema 1 (Chaitin) *Para toda palabra s (es decir, para toda $s \in 2^\star$)*

$$I(s) = H(s) + O(1)$$

Este resultado nos permite inferir que $I(s)$, la longitud del largo de programa para una palabra s, es una buena medida de la información que s contiene, ya que es "muy parecida" a su entropía ($H(s)$).

Nuestra mirada

Podemos pensar a la información como uuna entidad abstracta, pura, incorpórea, a la que sólo podemos acceder a través de representaciones.

Estas representaciones se diferencian entre sí por su estructura. Por ejemplo, tenemos por un lado *palabras*: agrupaciones ordenadas de símbolos, donde el orden juega un papel determinante (los entusiastas de los anagramas se divierten con esta propiedad). Por otro lado, tenemos *conjuntos*: colecciones de objetos donde no existe un orden entre ellos.

Cada una de estas estructuras (cuya diversidad obviamente no se agota en el par de ejemplos que vimos) define lo que llamaremos un *tipo de dato*.

Esta existencia de distintos tipos de datos no es caprichosa, ya que no cualquier tipo de dato es adecuado para expresar cualquier tipo de información. Por ejemplo, para una receta de cocina es mucho más útil tener los pasos que la conformen ordenados en una lista, ya que si estuvieran dispuestos en un conjunto (donde no habría una relación de orden entre ellos) podríamos estar echando los fideos a la olla antes de calentar el agua.

Dada esta diversidad de tipos de datos podríamos preguntarnos, ¿cómo cambian las teorías desarrolladas para palabras, cuando trabajamos con otros tipos de datos? Y, acaso antes que la anterior, ¿podemos utilizar esta maquinaria (intrínsecamente relacionada con las palabras) para movernos en otros mundos, donde habiten otros tipos de datos?

Estas preguntas condensan la luz del faro que guía este trabajo: extender las teorías que versan sobre palabras hacia otras más generales que se apliquen sobre otros objetos. En particular, buscaremos una noción de programa mínimo para un objeto y la relación con su entropía, expuesta por el teorema de la codificación.

Un primer paso en esa dirección fue dado por Chaitin en [2], cuando extiende su teoría de largo de programa para palabras en una que versa sobre conjuntos de naturales.

Un mundo nuevo

> De pronto parecieron estallar burbujas bajo mis párpados cerrados y mis manos se aferraron como garras a los obenques. Volví a la vida con un estremecimiento. Y, a sotavento, a mucho menos de cuarenta brazas, flotaba un gigantesco cachalote, como el casco volcado de una fragata, resplandeciendo al sol como un espejo su ancho lomo. La ballena, ondulando perezosamente en el mar, y lanzando de cuando en cuando su surtidor, parecía un honrado burgués fumando su pipa.
>
> Herman Melville en *Moby Dick*

Melville no podía (me atrevo a suponer) poner una ballena en un libro, pero sí podía escribir palabras; y las usó, en su libro "Moby Dick", para ponernos frente a un animal que pocos tendremos la oportunidad de ver realmente. Ha fabricado, y lo sigue haciendo, en la cabeza de quienes leemos sus palabras la imagen de una ballena colosal.

Similarmente, nosotros no podríamos poner cualquier objeto en manos de nuestras máquinas, pero como la imposibilidad de Melville de poner ballenas en los libros no nos impide pensar en ballenas, que no podamos poner (por ejemplo)

conjuntos en las computadoras, no nos va a impedir que nuestras máquinas trabajen con ellos.

Nosotros también vamos a utilizar *descripciones* para los objetos que nos interese estudiar con estas máquinas. Estaremos *representando* los objetos que necesitemos a través de una función de representación.

Esta función será la encargada de unir dos mundos: el de los objetos concretos que nuestras máquinas pueden manipular (por ejemplo palabras escritas con ceros y unos) al que llamaremos $\mathscr{D}$ y al de los objetos abstractos que están fuera de su alcance, al que llamaremos $\mathscr{O}$.

Definición 1 (Función de representación)

$$\mathfrak{v} : \mathscr{D} \to \mathscr{O}$$

En nuestro ejemplo, $\mathfrak{v}$ no es otra cosa que la pericia de Melville que nos permite obtener la imagen de la ballena (que vive en $\mathscr{O}$) dada una sucesión de palabras escritas en un libro (que viven en el mundo concreto $\mathscr{D}$).

Aún a esta altura, temprana hora del desarrollo de este trabajo, uno puede comenzar a intuir que la elección de la función de representación, es decir las características de la fución que uno elija, van a tener un peso determinante en los resultados que puedan alcanzarse.

Pero, ¿cómo saber qué tan buena (o tan mala) es una función de representación?¿En qué propiedades deberíamos fijarnos?

Para tratar de orientar las respuestas a estas preguntas vamos a proponer enfocar la atención sobre ciertas cualidades de las funciones de este tipo de las que hablan Heintz et al. en [4]. Como ellos trabajan sobre el problema de buscar codificaciones de clases contínuas de objetos matemáticos (en particular: funciones polinomiales) y nosotros lo haremos sobre clases discretas, deberemos modificar levemente estas cualidades para que se adapten mejor a nuestros propósitos.

Definición 2 (Representación inyectiva) *Diremos que un conjunto de objetos $\mathscr{D}$, vía una función de representación $\mathfrak{v} : \mathscr{D} \to \mathscr{O}$, representa* inyectivamente *a un conjunto de objetos $\mathscr{O}$ si, por cada objeto abstracto $o \in \mathscr{O}$, existe sólo un elemento concreto $d \in \mathscr{D}$ tal que*

$$\mathfrak{v}(d) = o$$

Definición 3 (Identidad) *Sea $\mathfrak{v}$ una función de representación. Llamaremos $id_{\mathfrak{v}} : \mathscr{D} \times \mathscr{D}$ al predicado tal que, dadas dos representaciones $d_1, d_2 \in \mathscr{D}$, cumple que*

$$id_{\mathfrak{v}}(d_1, d_2) \text{ es verdadero sii } \mathfrak{v}(d_1) = \mathfrak{v}(d_2)$$

Por lo pronto, la única restricción que pondremos sobre las funciones de representación es que su dominio esté incluído en el codominio de nuestros dispositivos de cómputo (decidimos, quizás en un acto de miopía, no brindar interés a la interpretación de palabras que no podemos escribir).

Un mundo distinto

Ya en la extensión mencionada que realiza Chaitin en [2], comienzan a plantearse diferencias. No todos los resultados que valían en el caso de las palabras siguen valiendo cuando se trabaja con conjuntos de naturales.

Uno de los resultados que deja de valer es el *teorema de codificación*. Sobre él centraremos principalmente nuestra atención a lo largo del presente trabajo, tomándolo como referencia de los cambios que pueden ocurrir cuando trabajamos con distintos modelos de cómputo (cómputos finitos y eternos) y distintos tipos de datos.

En la primera parte nos ocuparemos de las máquinas de cómputo finito y veremos un ejemplo de representación no trivial (trabajando con conjuntos finitos de naturales) en el que se verifica el teorema de codificación.

Luego mostraremos condiciones suficientes para la validez del teorema de invarianza en este modelo de cómputo.

Teorema 2 *Sea una función de representación* $\mathfrak{v} : \mathscr{D} \to \mathscr{O}$ *y su predicado de identidad asociado* $id_{\mathfrak{v}} : \mathscr{D} \times \mathscr{D}$. *Bajo el modelo de cómputo determinado por las máquinas de cómputos finitos, si se cumple cualquiera de estos items*

1. $\mathfrak{v}$ *total,* $id_{\mathfrak{v}}$ *recursivo y total*
2. $\mathfrak{v}$ *parcial,* $id_{\mathfrak{v}}$ *recursivo y total*
3. $\mathfrak{v}$ *parcial con dominio recursivo,* $id_{\mathfrak{v}}$ *recursivo parcial*

para todo $d \in \mathscr{D}$, *tal que* $d \in Dom(\mathfrak{v})$,

$$I(\mathfrak{v}(d)) = H(\mathfrak{v}(d)) + O(1)$$

En la segunda parte nos ocuparemos de las máquinas que son capaces de realizar cómputos eternos y daremos una pequeña extensión a algunos resultados que Chaitin muestra en [2] y que nos permiten ver cómo se ve afectado el teorema de la codificación al trabajar con conjuntos de naturales.

Además daremos condiciones suficientes sobre $\mathscr{A}$, una clase de conjuntos de naturales, para la validez del mencionado teorema bajo este modelo de cómputo. Para ello utilizaremos tres funciones: una función de *representaciones canónicas* $\mathbf{e} : \mathscr{A} \to 2^\star$, una función $\mathbf{c_e} : 2^\star \to 2^\star$ que nos permita adivinar una representación canónica de un conjunto dada una parte (finita) de cualquiera de sus representaciones (vía $\mathfrak{v}$), y una función $\mathbf{a} : 2^\star \to 2^\omega$ que dada una representación canónica nos indique el conjunto codificado por ella.

Teorema 3 *Sea* $\mathscr{A}$ *una clase de conjuntos de naturales para la cual puedan definirse*

$\mathbf{e} : \mathscr{A} \to 2^\star$ *una función inyeciva total*

$\mathbf{c_e} : 2^\star \to 2^\star$ *una función tal que, para todos* $s, t \in 2^\star$, *si* $\mathbf{c_e}(s) = e(A)$,

$$\mathbf{c_e}(st) \in Img(\mathbf{e}) \Rightarrow \mathbf{c_e}(st) = e(A)$$

$\mathbf{a} : 2^\star \to 2^\omega$ *una función tal que, para todo* $A \in \mathscr{A}$,$\mathfrak{v}(\mathbf{a}(\mathbf{e}(A))) = A$

Si $\mathbf{c_e}$ *y* $\mathbf{e}$ *son recursivas, existe un natural* $c_{\mathscr{A}}$ *tal que, para todo* $A \in \mathscr{A}$,

$$I^\infty(A) \leq H^\infty(A) + c_{\mathscr{A}}$$

Parte I

Cómputos finitos

Capítulo 1

Notas previas y definiciones

Todos los procesos que permitiremos sobre los objetos (secuencias[1], conjuntos) con los que trabajaremos a lo largo de este trabajo no podrán cruzar la frontera de disponer sólo de medios finitos. Siguiendo a *Turing*[1], enarbolaremos la justificación de esta restricción en el hecho de que la memoria humana es necesariamente limitada.

En particular, para las máquinas que describiremos en esta sección, también exigiremos que el tiempo que tarden en realizar su tarea sea, a su vez, finito.

1.1. Máquinas para cadenas

Podemos entender a nuestras máquinas como dispositivos (obviamente) imaginarios a los que le entregamos datos y nos devuelve cierta información[2]: simplemente una máquina procesadora de datos.

Como estamos tratando de capturar el mecanismo mental que permite a un ser humano realizar una tarea monótona, le brindaremos a estas máquinas la capacidad de mantener sólo un estado interno por vez, pero también la de ir mutando entre una cantidad finita de ellos. Serán estos estados internos los que que determinarán su comportamiento.

Así como nosotros nos ayudamos en nuestras tareas leyendo y escribiendo de y en papeles, por ejemplo, dotaremos a nuestras máquinas con cintas divididas en cuadros, donde cada uno de ellos sólo podrá contener un 1, un 0, o bien estar vacío.

Leerá los datos de entrada para su proceso de una cinta que llamaremos *fuente*, pero no le será posible escribir sobre ella. Esta cinta no contendrá espacios en blanco. Escribirá sus resultados sobre una cinta que llamaremos *destino*,

[1] Siempre que digamos *secuencia*, nos estaremos refiriendo a una sucesión infinita, generalmente de símbolos.

[2] De hecho, se suele pensar a estas máquinas [3] como dispositivos decodificadores: dado un mensaje, éste es decodificado a una forma que sea fácil de interpretar por el receptor.

de la cual no podrá borrar lo escrito. También contará con una cinta *de trabajo*, en la que sí permitiremos escribir y borrar libremente.

Estas cintas tienen un comienzo, pero no tienen fin. En el inicio del proceso, nuestra máquina se encuentra leyendo el comienzo (es decir, el primer cuadro) de cada una de las cintas.

Como ya mencionamos, la cualidad que más nos interesará de estas máquinas es su capacidad de "avisar" cuando han terminado su trabajo, al alcanzar alguno de sus *estados finales*, estados internos particulares.

Para considerar que ha realizado una *computación exitosa*, requeriremos que alcance alguno de sus estados finales. En ese momento, podremos leer el resultado de su operación desde su cinta de resultados. En ese caso, diremos que la computación está *definida*. En caso contrario diremos que está *indefinida*.

Así, para obtener un cómputo exitoso con estas máquinas, el tiempo que lleve este proceso debe ser finito. Es evidente, entonces, que nunca leerá todo el contenido de la cinta fuente.

Definición 4 (*Chaitin* ▷ Programa) *Llamaremos* programa *a la cadena formada por los dígitos de la cinta de entrada que lee la computadora durante un proceso de cómputo exitoso.*

A través de los programas estas máquinas reciben los datos necesarios para realizar sus procesos de cómputo. En este sentido, los programas conforman la *entrada de datos* de estos dispositivos.

Siguiendo el modelo que presenta Chaitin en [3], incluiremos una restricción más a estas computadoras: el conjunto de programas para cada máquina debe ser libre de prefijos. En otras palabras: si p es un programa, ninguno de sus prefijos puede serlo.

Definición 5 (*Chaitin* ▷ Máquinas de cómputos finitos para cadenas) *Llamaremos* máquina de cómputo finito para cadenas *a las máquinas descriptas en esta sección, y nos referiremos a ellas utilizando letras mayúsculas cursivas (como por ejemplo $\mathscr{M}$).*

Notaremos $\mathscr{M}(p) = s$ para indicar que, ante una computación exitosa, una computadora $\mathscr{M}$ lee un programa p y emite una cadena s como resultado. Como habíamos mencionados, estos aparatos pueden verse como funciones de $2^\star$ en $2^\star$.

1.2. Máquinas para conjuntos

Las cadenas impresas por las máquinas que acabamos de definir pueden verse como la representaciones de conjuntos finitos, por ejemplo de la manera que describimos a continuación.

Definición 6 (*Chaitin* ▷ Representación de conjuntos sobre cadenas) *Para representar conjuntos finitos de naturales con cadenas de ceros y unos utilizaremos la función $\mathfrak{v} : 2^\star \to \mathscr{P}_{\text{FIN}}(\mathbb{N}_0)$ tal que, para todo $s \in 2^\star$ y para todo natural a, $a \in \mathfrak{v}(s)$ sii hay a ceros entre dos unos en s.*

Tal como puede verse, la falta de inyectividad de esta función provoca que cada conjunto finito tenga infinitas cadenas que lo representan.

No debería resultar extraña la presencia de esta propiedad, ya que no sería esperable que pudiéramos escapar a este hecho al tratar de interpretar un tipo de dato evidentemente ordenado (como las cadenas de símbolos) en objetos donde el orden deja de importar, como en los conjuntos.

Definición 7 (Máquinas de cómputos finitos para conjuntos) *Nos interesará, entonces, interpretar los resultados de las máquinas definidas en la sección anterior, como conjuntos de naturales. Lo haremos aplicando la función de representación* $\mathfrak{v}$ *sobre el resultado de una máquina para cadenas.*
Al referirnos a este tipo de máquinas utilizaremos letras góticas mayúsculas (como por ejemplo $\mathfrak{M}$*).*

Notemos que, así como hemos observado que las máquinas de cómputos finitos para palabras $\mathscr{M}$ pueden verse como funciones $\mathscr{M} : 2^\star \to 2^\star$, las máquinas para conjuntos $\mathfrak{M}$ pueden verse como funciones $\mathfrak{M} : 2^\star \to \mathscr{P}_{\text{FIN}}(\mathbb{N}_0)$ conformadas por la composición entre la función de representación y una computadora para palabras, es decir

$$\mathfrak{M} = \mathfrak{v} \circ \mathscr{M}$$

Así, por cada máquina para conjuntos $\mathfrak{M}$ tenemos una computadora para palabras $\mathscr{M}$ subyacente a ella. Con este espíritu, nos tomaremos la licencia de decir que $\mathfrak{M}$ imprime un conjunto cada vez que su máquina subyacente $\mathscr{M}$ imprima una representación (vía $\mathfrak{v}$) para él.

1.3. Máquinas universales

En esta y en próximas secciones nos interesará referirnos a máquinas de cómputo finito sin que nos interese el tipo de resultado que emitan. En esos casos, las nombraremos con letras mayúsculas de imprenta (como por ejemplo M).

Definición 8 (*Chaitin* $\triangleright$ Máquina universal) *Una* máquina universal de cómputo finito *es una máquina de cómputo finito* U *tal que para cada máquina de cómputo finito* M *existe una constante* c_M *que cumple que: si* $M(p)$ *está definida, hay un programa* q *tal que* $U(q) = M(p)$ *y* $|q| \leq |p| + c_M$*.*

Teorema 4 (Chaitin) *Existe una máquina universal de cómputos finitos.*

Demostración. Hemos marcado más arriba que cada máquina puede (gracias a tener una cantidad finita de estados internos y manejar conjuntos finitos de símbolos de lectura y escritura) resumir su comportamiento en una tabla (finita) que indique, por cada estado posible de la máquina, la acción que debe realizar y el estado interno al que evoluciona.

La cantidad de tablas de este tipo que pueden construirse, y por lo tanto la cantidad de comportamientos distintos, es numerable. Luego, dada una

enumeración de estas tablas que pueda ser confeccionada por una máquina de cómputo finito, veamos como podemos usarla para determinar una máquina universal.

Sea una máquina que lee su cinta de entrada hasta hallar el primer 1. Si ha leído i ceros, simula el comportamiento de la máquina determinada por la i-ésima tabla en la enumeración mencionada.

Entonces, el resultado que emitirá su computación coincidirá con el que se consigue de la i-ésima computadora, escribiendo en su cinta de entrada el programa que resulta de quitar los primeros $i+1$ dígitos del programa original. Esto indica que la máquina descripta es una máquina universal, donde la constante de la que hablábamos en las definiciones es $i + 1$ para la i-ésima computadora.

Esta manera de indicar qué computadora debe ejecutarse nos asegura un dominio libre de prefijos para nuestra máquina universal. *Q.E.D.*

En adelante, al menos que se indique expresamente lo contrario, cuando hablemos de máquinas universales de cómputos finitos nos estaremos refiriendo a cualquiera de estas máquinas, que reconocen qué máquina particular deben imitar inspeccionando los prefijos de forma 0^i1 de sus programas.

1.4. Definiciones fundamentales

En esta sección presentaremos una batería de definiciones establecidas por Chaitin en [3] para máquinas sobre palabras y que extenderemos para el caso de los cómputos sobre conjuntos.

Programa minimal

Definición 9 (*Chaitin* ▷ Programa minimal) *Llamaremos* programas minimales para una palabra (repectivamente un conjunto) o en M, *notados* $o^\star_M$, *a los programas de menor longitud que provoquen que la máquina de cómputo finito* M *imprima* o *como resultado.*

Es claro que esta definición, así como ocurrirá con las siguientes, sólo tiene sentido para los casos en los que exista un programa para el elemento correspondiente en la máquina en cuestión.

Notemos que $o^\star_M$ podría denotar más de un programa. Sin embargo, en breve aparecerá en lugares donde esperaríamos que haya sólo un programa. En esos casos sólo importará hacer referencia a la cualidad de minimal de los programas, por lo que nos bastará con tomar cualquier representante (por ejemplo, el primero en orden lexicográfico y longitud).

Información algorítmica

Definición 10 (*Chaitin* ▷ Información algorítmica) *Llamaremos* información algorítmica de o en M *a la longitud de los programas minimales para* o *en* M. *La notaremos* $I_M(o)$.

Notemos que el programa minimal para un conjunto A es menor o igual, en tamaño, a los programas minimales de cualesquiera de sus representaciones.

Proposición 5 *Para todo s, si $\mathfrak{v}(s) = A$, entonces*

$$I_{\mathfrak{M}}(A) \leq I_{\mathscr{M}}(s)$$

A menudo nos estaremos refiriendo a propiedades que valen sobre computadoras universales. En esos casos notaremos $I(o)$, omitiendo el subíndice.

Probabilidad algorítmica

La siguiente definición puede verse intuitivamente como la probabilidad de obtener cierta palabra (o conjunto) como resultado de un proceso de cómputo en el cual la cinta fuente posee símbolos que se obtienen aleatoriamente.

Definición 11 (*Chaitin* $\triangleright$ Probabilidad algorítmica)

$$P_M(o) = \sum_{M(p)=o} 2^{-|p|}$$

De esta definición, podemos deducir trivialmente la siguiente proposición, que nos muestra que la probabilidad de imprimir un conjunto es la probabilidad de imprimir cualquiera de sus representaciones.

Proposición 6 *Para todo $A \in \mathscr{P}_{\text{FIN}}(\mathbb{N}_0)$,*

$$P_{\mathfrak{M}}(A) = \sum_{s|\mathfrak{v}(s)=A} P_{\mathscr{M}}(s)$$

Para ver que esta definición cumple las propiedades de una probabilidad nos basta con lo siguiente: Chaitin muestra en [3] que, para toda palabra $s \in 2^\star, 0 \leq P_{\mathscr{M}}(s) \leq 1$ y que el mismo resultado puede demostrarse para cómputos sobre conjuntos utilizando la propiedad anterior y que $\sum_{s\in 2^\star} P_{\mathscr{M}}(s) \leq 1$ (ver [3]).

Igual que antes, utilizaremos $P(o)$ para los casos en los que intervengan máquinas universales.

Entropía algorítmica

Definición 12 (*Chaitin* $\triangleright$ Entropía algorítmica)

$$H_M(o) = -\log P_M(o)$$

Sorpresivamente, nos conformaremos con escribir $H(o)$ cuando trabajemos con máquinas universales.

Cota superior para la entropía

Un resultado obvio, ya que la longitud de los programas minimales es el aporte más grande para el cálculo de la probabilidad, nos muestra una cota superior para la entropía de los objetos que estamos estudiando.

Proposición 7 (Chaitin) *Para toda cadena (respectivamente conjunto) o, vale que*

$$H(o) \leq I(o)$$

Optimalidad de las máquinas universales

Las máquinas universales son particularmente interesantes, y de hecho, son las que más usaremos en el transcurso de este trabajo, gracias a la cualidad de optimalidad que ostentan.

Esta cualidad se pone de manifiesto en los siguientes resultados que suelen engoblarse bajo el nombre de *teorema de invarianza*.

Teorema 8 (*Chaitin* $\triangleright$ Teorema de invarianza) *Para toda computadora de cómputos finitos M existe un natural c_M tal que, para toda cadena (respectivamente conjunto) o, vale que*

1. $I_U(o) \leq I_M(o) + c_M$
2. $P_M(o) \cdot 2^{-c_M} \leq P_U(o)$
3. $H_U(o) \leq H_M(o) + c_M$

Demostración. En [3, Teorema 2.3 (pág. 8)], Chaitin nos muestra que

$$I_{\mathscr{U}}(s) \leq I_{\mathscr{M}}(s) + c_{\mathscr{M}}$$

Si tenemos un programa minimal para un conjunto A, tenemos un programa minimal para alguna de sus representaciones. Llamémosle s_A. Usando el resultado que acabamos de ver, sabemos que $I_{\mathscr{U}}(s_A) \leq I_{\mathscr{M}}(s_A) + c_{\mathscr{M}}$ y, como hemos notado más arriba (Proposición 5), $I_{\mathscr{U}}(A) \leq I_{\mathscr{U}}(s_A)$. Luego,

$$I_{\mathfrak{U}}(A) \leq I_{\mathfrak{M}}(A) + c_{\mathfrak{M}}$$

Así completamos la demostración del ítem 1.

Con respecto al item 2 notemos que, estemos trabajando con palabras o con conjuntos, definimos la probabilidad de que una máquina M imprima o como

$$P_M(o) = \sum_{p|M(p)=o} 2^{-|p|}$$

Por la definición que vimos de máquina universal, sabemos que existe un natural c_M tal que, por cada uno de esos programas p, existe un programa q_p^M

tal que $U(q_p^M) = M(p)$ y $|q_p^M| \leq |p| + c_M$. Entonces, si multiplicamos la ecuación anterior por 2^{-c_M},

$$P_M(o) \cdot 2^{-c_M} = \sum_{p|M(p)=o} 2^{-|p|-c_M} \leq \sum_{p|M(p)=o} 2^{-|q_p^M|}$$

Como acabamos de ver, cada uno de estos programas q_p^M provocan que U imprima o, pero podría haber otros programas q que provocasen el mismo resultado en los cómputos de U. Entonces

$$P_M(o) \cdot 2^{-c_M} \leq \sum_{q|U(q)=o} 2^{-|q|} = P_U(o)$$

Así se completa la demostración del inciso 2.

Finalmente, aplicando la definición de entropía que hemos establecido más arriba en el resultado anterior podemos obtener el ítem que faltaba.

$$H_U(o) \leq H_M(o) + c_M$$

Es importante notar que en estas dos últimas observaciones no tiene importancia el tipo de dato de nuestro resultado. *Q.E.D.*

Capítulo 2

Teorema de la codificación

Uno de los principales resultados que muestra Chaitin en su *teoría de largo de programa* [3], es aquel que nos indica que la compresibilidad de una cadena (es decir, la diferencia entre la longitud de la cadena y la longitud de un programa mínimo para la cadena) es una buena medida de la información que la cadena contiene, en cuanto se corresponde con el concepto de información que estableciera Shannon.

Toda esta teoría de Chaitin se construye utilizando máquinas de cómputo finito. Al tratar de extenderla, migrando hacia las máquinas de cómputo eterno que emiten conjuntos de naturales, este resultado no se verifica para cualquier tipo de conjuntos.

Pero, ¿qué ocurrirá en medio de estos dos mundos?¿Podrá una teoría que se contruya sobre máquinas de cómputo finito pero que emitan conjuntos, ser idéntica (formalmente) a la teoría de la información de Shannon?

En este capítulo veremos que sí existe una teoría de este estilo. Previamente repasaremos el resultado mencionado para palabras.

2.1. Teorema de la codificación para palabras

Teorema 9 (*Chaitin* $\triangleright$ Teorema de la codificación para palabras) *Sea $\mathscr{M}$ a una máquina de cómputos finitos sobre palabras. Para toda cadena $s \in 2^\star$,*

$$I(s) = H_{\mathscr{M}}(s) + O(1)$$

Demostración. La demostración de este teorema se basa en la Proposición 7, y en el teorema siguiente, presentado por Chaitin en [3] y del cual presentaremos una versión reorganizada de la demostración que allí puede encontrarse.

Teorema 10 (Chaitin) *Para cada computadora $\mathscr{M}$, existe una constante m tal que, para toda cadena $s \in 2^\star$,*

$$I(s) \leq -\log P_{\mathscr{M}}(s) + m$$

Demostración. Vamos a mostrar que, para cada computadora $\mathscr{M}$, existe una máquina $\mathscr{M}_{\mathbf{T}}$ que cumple que, para todo $s \in 2^\star$,

$$I_{\mathscr{M}_{\mathbf{T}}}(s) \quad = \quad \lceil -\log P_{\mathscr{M}}(s) \rceil + 1 \tag{2.1}$$

Si existiese una máquina así, nos bastaría con utilizar el teorema de invarianza (8) para ver que

$$I_{\mathscr{U}}(s) \leq I_{\mathscr{M}_{\mathbf{T}}}(s) + c_{\mathscr{M}_{\mathbf{T}}} = \lceil -\log P_{\mathscr{M}}(s) \rceil + 1 + c_{\mathscr{M}_{\mathbf{T}}} \leq -\log P_{\mathscr{M}}(s) + 2 + c_{\mathscr{M}_{\mathbf{T}}}$$

que es lo que queremos demostrar (con $m = 2 + c_{\mathscr{M}_{\mathbf{T}}}$).

Una computadora a medida

Para demostrar la existencia de esta máquina $\mathscr{M}_{\mathbf{T}}$ necesitamos ver que somos capaces de construirnos una máquina que cumpla ciertos *requerimientos*.

En lugar de exigir directamente que esta máquina cumpla determinadas propiedades en cuanto a los tamaños mínimos de los programas para cada resultado, y sus probabilidades, impondremos requisitos un tanto más detallados: daremos una lista de parejas $\mathbf{T} = \{\langle s_k, n_k \rangle\}$ y exigiremos que, por cada una de ellas, exista un programa de tamaño n_k que provoque que la máquina emita el resultado s_k.

Obviamente, la máquina que cumpla estos requerimientos (llamémosla $\mathscr{M}_{\mathbf{T}}$) provocará que $P_{\mathscr{M}_{\mathbf{T}}}(s) = \sum_{s=s_k} 2^{-n_k}$ y $I_{\mathscr{M}_{\mathbf{T}}}(s) = \min_{s=s_k} n_k$.

Pero no cualquier lista de requerimientos de este estilo podrá ser satisfecha. Sólo podrán serlo aquellas que cumplan con la restricción que enuncia el siguiente teorema (cuya demostración puede verse en [3]).

Teorema 11 (Kraft-Chaitin) *Dado un conjunto de parejas* $\mathbf{T} = \{\langle s_k, n_k \rangle\}$ *posiblemente infinito, tal que* $\sum_k 2^{-n_k} \leq 1$ *(en cuyo caso lo llamaremos* consistente*), existe una máquina de cómputo finito* $\mathscr{M}_{\mathbf{T}}$ *tal que hay tantos programas de longitud* n *que provocan que* $\mathscr{M}_{\mathbf{T}}(p) = s$ *como pares* $\langle s, n \rangle$ *haya en* $\mathbf{T}$.

Aún más, si estos requerimientos son emitidos uno a uno, se puede simular una computadora que los satisface.

Requeriendo requerimientos

Veamos cómo podemos utilizar lo que acabamos de aprender para completar la demostración del teorema 10.

En principio, le pediremos a $\mathscr{M}_{\mathbf{T}}$ que construya una lista de requerimientos $\mathbf{T}$ y que simule la computadora que ellos determinen. Pero, ¿cómo deben ser estos requerimientos para que dicha máquina nos garantice las propiedades enunciadas más arriba (2.1)?

Naturalmente, por cada cadena s en la imagen de $\mathscr{M}$, $\langle s, \lceil -\log P_{\mathscr{M}}(s) \rceil + 1 \rangle$ debe estar entre nuestros requerimientos.

Sería ideal que sólo esos fueran nuestros requerimientos, pero nuestra computadora $\mathscr{M}_{\mathbf{T}}$ no será capaz de construir una lista de requerimientos de este tipo,

ya que una máquina de cómputo finito no puede conocer las probabilidades de otra, aunque sí puede aproximarlas.

Para que $\langle s, \lceil -\log P_{\mathscr{M}}(s)\rceil + 1\rangle$ esté en nuestra lista de requerimientos $\mathbf{T}$, nos basta con que $\mathbf{T}$ contenga sólo una pareja $\langle s, n\rangle$ por cada s y n que verifiquen que $n \geq \lceil -\log P_{\mathscr{M}}(s)\rceil + 1$ y ninguna para aquellas que no lo hagan. Entonces,

$$\begin{array}{lll} \langle s, n+1\rangle \in \mathbf{T} & \text{sii} & n \geq \lceil -\log P_{\mathscr{M}}(s)\rceil \\ & \text{sii} & n > -\log P_{\mathscr{M}}(s) \\ & \text{sii} & 2^{-n} < P_{\mathscr{M}}(s) \end{array}$$

Y esta lista de requerimientos sí puede producirla $\mathscr{M}_{\mathbf{T}}$, tal como lo dice el siguiente lema, que es una consecuencia obvia del hecho de que el dominio de $\mathscr{M}$ sea recursivamente enumerable[1].

Lema 12 (Chaitin) *El conjunto de proposiciones verdaderas de la forma*

$$\text{“}P_{\mathscr{M}}(s) > r\text{”}$$

es recursivamente enumerable.

Requerimientos de buena cuna

Sólo nos resta verificar que estos requerimientos son consistentes de acuerdo al teorema 11. Para ello, definamos los siguientes conjuntos.

$Img_{\mathscr{M}_{\mathbf{T}}}$ al conjunto de resultados que queremos que imprima $\mathscr{M}_{\mathbf{T}}$,

$Dom^{s}_{\mathscr{M}_{\mathbf{T}}}$ al conjunto de cadenas que queremos asignar al resultado s, y

$Dom_{\mathscr{M}_{\mathbf{T}}} = \bigcup_{s \in Img_{\mathscr{M}_{\mathbf{T}}}} Dom^{s}_{\mathscr{M}_{\mathbf{T}}}$.

Observemos lo siguiente:

$$\sum_{\langle t,n\rangle \in \mathbf{T}} 2^{-n} = \sum_{p \in Dom_{\mathscr{M}_{\mathbf{T}}}} 2^{-|p|} = \sum_{s \in Img_{\mathscr{M}_{\mathbf{T}}}} \left(\sum_{p \in Dom^{s}_{\mathscr{M}_{\mathbf{T}}}} 2^{-|p|} \right) \tag{2.2}$$

Luego, gracias a que, como vimos antes, $\langle s, m\rangle$ es tomado como requerimiento si y sólo si $m \geq \lceil -\log P_{\mathscr{M}}(s)\rceil + 1$, la cantidad de programas p de longitud m tal que $\mathscr{M}_{\mathbf{T}}(p) = s$ es 1 si m es mayor o igual que $\lceil -\log P_{\mathscr{M}}(s)\rceil + 1$, y cero en otro caso. Justamente el objetivo de esta construcción es que haya exactamente un programa mínimo para cada resultado s de $\mathscr{M}$ y a lo sumo un programa de cada longitud mayor a la del mínimo que también imprima s.

Entonces, para cada s en $Img_{\mathscr{M}_{\mathbf{T}}}$, vale que

$$\sum_{p \in Dom^{s}_{\mathscr{M}_{\mathbf{T}}}} 2^{-|p|} = 2^{-\lceil -\log P_{\mathscr{M}}(s)\rceil} \leq P_{\mathscr{M}}(s) \tag{2.3}$$

[1] Recordemos que un conjunto es *recursivamente enumerable* cuando existe un procedimiento que lista sus elementos.

Utilizando (2.2), (2.3), que $Img_{\mathscr{M}_{\mathbf{T}}} \subseteq Img(\mathscr{M})$, conseguimos

$$\sum_{\langle s,n\rangle\in\mathbf{T}} 2^{-n} \leq \sum_{s\in Img_{\mathscr{M}_{\mathbf{T}}}} P_{\mathscr{M}}(s) \leq \sum_{s\in Img(\mathscr{M})} P_{\mathscr{M}}(s)$$

Finalmente, nos apoyaremos en el siguiente lema.

Lema 13 (Chaitin) *Sea $\mathscr{M}$ una máquina de cómputo finito. Para toda palabra s en $2^\star$*

$$\sum_{s\in Img(\mathscr{M})} P_{\mathscr{M}}(s) \leq 1$$

Uniéndolo con la inecuación anterior concluimos la verificación de la consistencia de los requerimientos $\mathbf{T}$, y con ella la demostración del teorema 10. *Q.E.D.*

2.2. Teorema de la codificación para conjuntos

Teorema 14 (Teorema de la codificación para conjuntos) *Sea $\mathfrak{M}$ una máquina de cómputos finitos sobre conjuntos. Para todo conjunto $A \in \mathscr{P}_{\text{FIN}}(\mathbb{N}_0)$,*

$$I(A) = H_{\mathfrak{M}}(A) + O(1)$$

Demostración. Basaremos la demostración de este teorema en la Proposición 7 y en una versión de la desigualdad enunciada en el teorema 10 pero para el caso en que los resultados de nuestras máquinas de cómputo finito sean interpretados como conjuntos de naturales.

Teorema 15 *Para cada computadora $\mathfrak{M}$ existe una constante m tal que, para todo $A \in \mathscr{P}_{\text{FIN}}(\mathbb{N}_0)$*

$$I(A) \leq -\log P_{\mathfrak{M}}(A) + m$$

Demostración. Usaremos la misma estrategia que en el teorema 10, para verificar que, por cada computadora $\mathfrak{M}$, existe una computadora $\mathfrak{M}_{\mathbf{T}_\mathfrak{v}}$ tal que

$$I_{\mathfrak{M}_{\mathbf{T}_\mathfrak{v}}}(A) = \lceil -\log P_{\mathfrak{M}}(A)\rceil + 1 \tag{2.4}$$

Como antes, de obtener una máquina con estas características, podríamos deducir, utilizando el teorema de invarianza (8)

$$I_{\mathfrak{U}}(A) \leq I_{\mathfrak{M}_{\mathbf{T}_\mathfrak{v}}}(A) + c_{\mathfrak{M}_{\mathbf{T}_\mathfrak{v}}} = \lceil -\log P_{\mathfrak{M}}(A)\rceil + 1 + c_{\mathfrak{M}_{\mathbf{T}_\mathfrak{v}}} \leq -\log P_{\mathfrak{M}}(A) + 2 + c_{\mathfrak{M}_{\mathbf{T}_\mathfrak{v}}}$$

que es lo que queremos demostrar (con $m = 2 + c_{\mathfrak{M}_{\mathbf{T}_\mathfrak{v}}}$).

Los requerimientos

Nuevamente, $\mathfrak{M}_{\mathbf{T}_\mathfrak{v}}$ construirá una lista de requerimientos $\mathbf{T}_\mathfrak{v}$ y simulará la máquina que ellos determinan.

Esta vez necesitamos que, por cada conjunto A en la imagen de $\mathfrak{M}$, $\mathbf{T}_\mathfrak{v}$ contenga a una (y sólo una) pareja $\langle s, \lceil H_\mathfrak{M}(\mathfrak{v}(s))\rceil + 1\rangle$ tal que $\mathfrak{v}(s) = A$.

Nos basta, entonces, con que $\mathbf{T}_\mathfrak{v}$ contenga, por cada conjunto finito de naturales A en el dominio de $\mathfrak{M}$, sólo una pareja $\langle s, n\rangle$ por cada s y n tales que $\mathfrak{v}(s) = A$ y $n \geq \lceil H_\mathfrak{M}(\mathfrak{v}(s))\rceil + 1$ y ninguna para aquellas que no lo cumplan.

Llamemos $\Pi_1^A(\mathbf{T}_\mathfrak{v})$ al conjunto de palabras s tales que $\mathfrak{v}(s) = A$ y existe un natural n que forma una pareja $\langle s, n\rangle$ que pertenece a $\mathbf{T}_\mathfrak{v}$. Dicho de otro modo, $\Pi_1^A(\mathbf{T}_\mathfrak{v})$ es un filtro sobre $\mathbf{T}_\mathfrak{v}$ que nos permite quedarnos con todas las cadenas que participan en requerimientos que refiren al conjunto A.

Entonces,

$$\begin{array}{lll} \langle s, n+1\rangle \in \mathbf{T}_\mathfrak{v} & \text{sii} & n \geq \lceil H_\mathfrak{M}(\mathfrak{v}(s))\rceil \text{ y } \Pi_1^{\mathfrak{v}(s)}(\mathbf{T}_\mathfrak{v}) = \{s\} \\ & \text{sii} & n > H_\mathfrak{M}(\mathfrak{v}(s)) \text{ y } \Pi_1^{\mathfrak{v}(s)}(\mathbf{T}_\mathfrak{v}) = \{s\} \\ & \text{sii} & 2^{-n} < P_\mathfrak{M}(\mathfrak{v}(s)) \text{ y } \Pi_1^{\mathfrak{v}(s)}(\mathbf{T}_\mathfrak{v}) = \{s\} \end{array}$$

Esta lista de requerimientos puede ser producida por una máquina de cómputos finitos ya que el dominio de $\mathfrak{M}$ es recursivamente enumerable y para aplicar el filtro $\Pi_1^A(\mathbf{T}_\mathfrak{v})$ sólo necesitamos poder determinar (recursivamente) si dos representaciones refieren a un mismo objeto; lo que es trivial para la función de representación con la que estamos trabajando ($\mathfrak{v}$).

Consistencia

La verificación de la consistencia de los requerimientos en $\mathbf{T}_\mathfrak{v}$ es muy similar a la de los requerimientos en $\mathbf{T}$. Los únicos detalles dignos de mención son los siguientes.

En primer lugar, en la ecuación (2.3), lo que nos permite decir que, para cada $A \in Img_{\mathfrak{M}_{\mathbf{T}_\mathfrak{v}}}$, vale que

$$\sum_{p \in Dom^A_{\mathfrak{M}_{\mathbf{T}_\mathfrak{v}}}} 2^{-|p|} = 2^{-\lceil -\log P_\mathfrak{M}(A)\rceil} \leq P_\mathfrak{M}(A)$$

es la forma en la que armamos nuestra lista de requerimientos, entre cuyas características no podemos despreciar el filtro Π_1^A.

En segundo lugar, el lema 13 es fácilmente extensible a conjuntos como se muestra a continuación.

Lema 16 *Sea $\mathscr{M}$ una máquina de cómputo finito cualquiera.*

$$\sum_{A \in \mathscr{P}_{\mathrm{FIN}}(\mathbb{N}_0)} P_\mathfrak{M}(A) \leq 1$$

Demostración. Como $\sum_{A\in\mathscr{P}_{\text{FIN}}(\mathbb{N}_0)} P_{\mathfrak{M}}(A) = \sum_{A\in\mathscr{P}_{\text{FIN}}(\mathbb{N}_0)}(\sum_{s|\mathfrak{v}(s)=A} P_{\mathscr{M}}(s))$, y ya que el dominio de $\mathfrak{v}$ está incluído en $2^\star$, la sumatoria del enunciado de este lema no puede ser mayor que $\sum_{s\in 2^\star} P_{\mathscr{M}}(s)$. Finalmente, sabemos que dicha cantidad (por lema 13) no es mayor a uno. *Q.E.D.*

Esto concluye la demostración del teorema 15. *Q.E.D.*

El del estribo

El siguiente corolario nos muestra que el aporte más significativo a la probabilidad de imprimir un conjunto lo brinda el programa más corto que genera alguna de sus representaciones.

Corolario 17 *Llamemos s_A a alguna de las representacions más cortas de A. Existe una constante $r \in \mathbb{Q}$ tal que, para todo $A \in \mathscr{P}_{\text{FIN}}(\mathbb{N}_0)$,*

$$P(A) = rP(s_A).$$

Demostración. Observemos que $2^{-I(A)} = 2^{-I(s_A)}$. Por el teorema de codificación para conjuntos, sabemos que existe una constante c tal que $c \cdot P(A) = 2^{-I(A)}$. Entonces, $c \cdot P(A) = 2^{-I(s_A)}$.

Por otro lado, por teorema de invarianza (8), $2^{-I(s_A)} = d \cdot P(s_A)$ donde d es una constante.

Uniendo lo anterior, $c \cdot P(A) = d \cdot P(s_A)$, lo que nos conduce trivialmente al resultado que buscábamos, con $r = c/d$. *Q.E.D.*

Capítulo 3

Condiciones abstractas para el teorema de la codificación

Como mencionamos en la introducción, el que acabamos de ver es un mero ejemplo de trabajo con objetos abstractos: el caso particular de los conjuntos finitos de naturales.

Sin embargo, así como teníamos una función de representación

$$\mathfrak{v} : 2^\star \to \mathscr{P}_{\text{FIN}}(\mathbb{N}_0)$$

que une el mundo de las cadenas con el de los conjuntos finitos, podríamos utilizar otras funciones para trabajar con otros objetos.

El contraste entre las cualidades de estos objetos y la estructura de representación que utilicemos (por ejemplo el que hemos enfrentado entre conjuntos y cadenas en el capítulo anterior), determinará en la función de representación ciertas propiedades.

Estas propiedades, surgidas de la brecha a la que sirve de puente nuestra función de representación, pueden ser utilizardas para dar condiciones suficientes sobre la validez del teorema de la codificación.

En la presente sección utilizaremos las definiciones y notaciones establecidas en secciones anteriores para el trabajo sobre conjuntos, pero para referirnos a objetos abstractos cualesquiera. Así, por ejemplo, dada una máquina de cómputos finitos $\mathscr{M}$ llamaremos $\mathfrak{M}$ al dispositivo que resulta de aplicar la función de representación $\mathfrak{v}$ sobre $\mathscr{M}$, es decir, $\mathfrak{M}$ es la composición entre $\mathfrak{v}$ y $\mathscr{M}$; para todo programa p de $\mathscr{M}$,

$$\mathfrak{M}(p) = (\mathfrak{v} \circ \mathscr{M})(p)$$

Sólo haremos explícitas aquellas definiciones o características que sean relevantes.

Antes de continuar, recordemos el enunciado del teorema a cuya demostración dedicaremos el presente capítulo.

Teorema 2 *Sea una función de representación* $\mathfrak{v} : \mathscr{D} \to \mathscr{O}$ *y su predicado de identidad asociado* $id_{\mathfrak{v}} : \mathscr{D} \times \mathscr{D}$*. Bajo el modelo de cómputo determinado por las máquinas de cómputos finitos, si se cumple cualquiera de estos items*

1. $\mathfrak{v}$ *total,* $id_{\mathfrak{v}}$ *recursiva y total*
2. $\mathfrak{v}$ *parcial,* $id_{\mathfrak{v}}$ *recursiva y total*
3. $\mathfrak{v}$ *parcial con dominio recursivo,* $id_{\mathfrak{v}}$ *recursiva parcial*

Entonces, para todo $d \in \mathscr{D}$*, tal que* $d \in Dom(\mathfrak{v})$*,*

$$I(\mathfrak{v}(d)) = H(\mathfrak{v}(d)) + O(1)$$

Demostración. Dividiremos la demostración de este teorema en dos casos: en el que trabajemos con funciones totales de representación y en el que lo hagamos con funciones parciales.

3.1. Representaciones totales

En los ejemplos que hemos visto en capítulos anteriores, las funciones de representación utilizadas eran totales. Comencemos por este tipo de funciones, y dejemos para la próxima sección el estudio del caso en el que no lo sean.

Teorema 18 *Sea una función total de representación* $\mathfrak{v} : \mathscr{D} \to \mathscr{O}$ *y su predicado de identidad asociado* $id_{\mathfrak{v}} : \mathscr{D} \times \mathscr{D}$*. Bajo el modelo de cómputo que determinan las máquinas de cómputo finito, si* $id_{\mathfrak{v}}$ *es* recursiva *se cumple que, para todo* $d \in \mathscr{D}$*,*

$$I(\mathfrak{v}(d)) = H(\mathfrak{v}(d)) + O(1)$$

Demostración. En principio, debemos notar que como el universo que queremos utilizar (es decir, los objetos abstractos sobre los que queremos predicar) es la imagen de nuestra función de representación $\mathfrak{v}$, y gracias a que su dominio debe estar incluído en la imagen de las máquinas que estamos utilizando, para cada objeto que queremos representar, habrá una representación que puede ser producida por una computación generada en base a un determinado programa.

Así, fácilmente llegamos a demostrar la primera de las desigualdades que completarán la demostración del teorema. Notemos que esta parte es un resultado que ya hemos visto para el caso particular de los conjuntos finitos de naturales (Proposición 7).

Lema 19 *Para todo* $o \in \mathscr{O}$

$$H(o) \leq I(o)$$

Ahora, demostremos la otra desigualdad. Para ello vamos a utilizar el siguiente teorema.

Teorema 20 *Sea una función total de representación* $\mathfrak{v} : \mathscr{D} \to \mathscr{O}$ *y su predicado de identidad asociado* $id_{\mathfrak{v}} : \mathscr{D} \times \mathscr{D}$*. Si* $id_{\mathfrak{v}}$ *es* recursivo*, para cada máquina* $\mathfrak{M}$ *existe una constante* m *tal que, para todo* $o \in \mathscr{O}$*,*

$$I(o) \leq -\log P_{\mathfrak{M}}(o) + m$$

Demostración. Esta demostración es casi idéntica a la del teorema 15, ya que la estrategia seguida en dicha demostración es una especialización (en la que no se pierde generalidad) de otra más general, con la que se puede demostrar este teorema.

Sólo se debe reemplazar $\mathscr{P}_{\text{FIN}}(\mathbb{N}_0)$ por $\mathscr{O}$ y, donde se habla de conjuntos de números, pensar en objetos abstractos cualesquiera. *Q.E.D.*

Con este teorema se completa la demostración del teorema 18. *Q.E.D.*

Corolario 21 *Sea una función total de representación* $\mathfrak{v} : \mathscr{D} \to \mathscr{O}$ *y su predicado de identidad asociado* $id_{\mathfrak{v}} : \mathscr{D} \times \mathscr{D}$*. Bajo el modelo de cómputo que determinan las máquinas de cómputo finito, si* $\mathfrak{v}$ *es* inyectiva*, se cumple que, para todo* $d \in \mathscr{D}$*,*

$$I(\mathfrak{v}(d)) = H(\mathfrak{v}(d)) + O(1)$$

Este corolario es una consecuencia trivial del teorema 2, ya que si la función de representación es inyectiva, el predicado asociado a ella $id_{\mathfrak{v}}$ es obviamente recursivo, pues sólo consiste en la comparación sintática de sus argumentos.

Así como, obviamente, el teorema de codificación para conjuntos (teorema 15) era un ejemplo de la aplicación del teorema 2, el teorema 10 puede verse como un caso particular del corolario anterior.

3.2. Representación parcial

Hasta ahora hemos trabajado sólo con funciones de representación totales. Como era de esperarse, si liberamos esta restricción (es decir, si utilizamos una función de representación $\mathfrak{v} : 2^\star \to \mathscr{O}$ tal que existe una cadena $s \in 2^\star$ que no representa ningún objeto en $\mathscr{O}$), deberemos preocuparnos por ciertos detalles que antes no nos molestaban.

En principo, las máquinas $\mathfrak{M}$ sufren un cambio. Ahora no cualquier resultado de $\mathscr{M}$ podrá ser interpretado como un valor del universo abstracto $\mathscr{O}$, y por tanto, podemos pensar que $\mathfrak{M}$ tiene dos tipos de resultado: *resultados válidos* que son objetos en $\mathscr{O}$, y *residuos*, es decir, elementos que no pertenecen a $\mathscr{O}$. En ese sentido, definamos

Definición 13 (Imagen de $\mathfrak{M}$) *Sea* $\mathscr{M}$ *una máquina de cómputos finitos*

$$Img(\mathfrak{M}) = \{\mathfrak{v}(\mathscr{M}(p)) : \mathscr{M}(p) \in Dom(\mathfrak{v})\}$$

Definición 14 (Residuo de $\mathfrak{M}$) *Sea* $\mathscr{M}$ *una máquina de cómputos finitos*

$$Res(\mathfrak{M}) = \{\mathscr{M}(p) : \mathscr{M}(p) \notin Dom(\mathfrak{v})\}$$

Ahora sí, completemos la demostración del teorema 2 con el siguiente resultado.

Teorema 22 *Sea una función parcial de representación $\mathfrak{v} : \mathscr{D} \to \mathscr{O}$ y su predicado de identidad asociado $id_{\mathfrak{v}} : \mathscr{D} \times \mathscr{D}$. Bajo el modelo de cómputo que determinan las máquinas de cómputo finito, si se cumple cualquiera de estos items*

1. *$id_{\mathfrak{v}}$ es recursiva y total*

2. *$id_{\mathfrak{v}}$ es recursiva parcial y $\mathfrak{v}$ tiene dominio recursivo*

se cumple que, para todo $d \in \mathscr{D}$,

$$I(\mathfrak{v}(d)) = H(\mathfrak{v}(d)) + O(1)$$

Demostración. Utilizaremos nuevamente el lema 19 para mostrar una de las desigualdades de este teorema.

Para la otra, demostraremos el siguiente teorema

Teorema 23 *Sea una función parcial de representación $\mathfrak{v} : \mathscr{D} \to \mathscr{O}$ y su predicado de identidad asociado $id_{\mathfrak{v}} : \mathscr{D} \times \mathscr{D}$. Si se cumple cualquiera de estos items*

1. *$id_{\mathfrak{v}}$ es recursiva y total*

2. *$id_{\mathfrak{v}}$ es recursiva parcial y $\mathfrak{v}$ tiene dominio recursivo*

entonces, para todo $d \in \mathscr{D}$, tal que $d \in Dom(\mathfrak{v})$,

$$I(\mathfrak{v}(d)) \leq H(\mathfrak{v}(d)) + O(1)$$

Demostración. Seguiremos la estrategia del teorema 20, pero esta vez debemos tener cuidado con los residuos de $\mathfrak{M}$.

Vamos a mostrar que, para cada computadora $\mathscr{M}$, existe una máquina $\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}$ que cumple que, para todo $s \in Dom(\mathfrak{v})$,

$$I_{\mathfrak{M}_{\mathbf{R}_{\mathfrak{v}}}}(\mathfrak{v}(s)) = \lceil -\log P_{\mathfrak{M}}(\mathfrak{v}(s)) \rceil + 1 \tag{3.1}$$

Para ello utilizaremos la siguiente lista de requerimientos:

$$\mathbf{R}_{\mathfrak{v}} = \left\{ \langle s, n \rangle \;\middle|\; \begin{array}{l} s \notin Dom(\mathfrak{v}), P_{\mathscr{M}}(s) > 2^{-n} \\ s \in Dom(\mathfrak{v}), P_{\mathfrak{M}}(\mathfrak{v}(s)) > 2^{-n} \text{ y } \Pi_1^{\mathfrak{v}(s)}(\mathbf{R}_{\mathfrak{v}}) = \{s\} \end{array} \right\}$$

donde $\Pi_1^{\mathfrak{v}(s)}$ es el filtro que ya utilizamos en el teorema 15.

Factibilidad

Veamos, primero, que esta lista de requerimientos puede ser generada por una máquina de cómputos finitos.

En primer lugar, el dominio de $\mathscr{M}$ es recursivamente enumerable.

Luego, si $id_{\mathfrak{v}}$ es total, sólo nos dará *verdadero* si ambas representaciones (sus argumentos) pertenecen al dominio de $\mathfrak{v}$ y representan al mismo elemento, por tanto cada residuo distinto será acumulado como un nuevo requerimiento. Por otro lado, si $\mathfrak{v}$ tiene dominio recursivo, nuestra máquina puede hacer por sí misma el trabajo de dividir los resultados entre residuos y representaciones válidas.

En tercer lugar, para aplicar el filtro $\Pi_1^{\mathfrak{v}(s)}$, tenemos el predicado $id_{\mathfrak{v}}$.

Estos tres detalles nos muestran que $\mathbf{R}_{\mathfrak{v}}$ puede ser generada por una máquina de cómputos finitos.

Consistencia

Para ver la consistencia de estos requerimientos, definamos:

$Img_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}$ al conjunto de palabras que queremos que imprima $\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}$,

$Dom^s_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}$ al conjunto de cadenas que queremos asignar al resultado s, y

$Dom_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}} = \bigcup_{s\in Img_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}} Dom^s_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}$.

y veamos que

$$\sum_{\langle s,n\rangle\in\mathbf{R}_{\mathfrak{v}}} 2^{-n} = \sum_{p\in Dom_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}} 2^{-|p|} = \sum_{s\in Img_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}} \left(\sum_{p\in Dom^s_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}} 2^{-|p|} \right)$$

Como $\mathfrak{v}$ es parcial, tiene sentido separar la suma de las longitudes de los requerimientos de la siguiente manera:

$$\sum_{s\in(Img_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}} - Dom(\mathfrak{v}))} \left(\sum_{p\in Dom^s_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}} 2^{-|p|} \right) + \sum_{s\in(Img_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}} \cap Dom(\mathfrak{v}))} \left(\sum_{p\in Dom^s_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}} 2^{-|p|} \right)$$

Por otro lado, debido a la manera en la que armamos nuestra lista de requerimientos, obtenemos que para cada $s \in Img_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}$, vale que

$$\begin{array}{rcl} s \notin Dom(\mathfrak{v}) & \Rightarrow & \sum_{p\in Dom^s_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}} 2^{-|p|} = 2^{-\lceil -\log P_{\mathscr{M}}(s)\rceil} \leq P_{\mathscr{M}}(s) \\ s \in Dom(\mathfrak{v}) & \Rightarrow & \sum_{p\in Dom^s_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}}} 2^{-|p|} = 2^{-\lceil -\log P_{\mathfrak{M}}(\mathfrak{v}(s))\rceil} \leq P_{\mathfrak{M}}(\mathfrak{v}(s)) \end{array}$$

Juntándolo con lo anterior vemos que

$$\sum_{\langle s,n\rangle\in\mathbf{R}_{\mathfrak{v}}} 2^{-n} \leq \sum_{s\in(Img_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}} - Dom(\mathfrak{v}))} P_{\mathscr{M}}(s) + \sum_{s\in(Img_{\mathscr{M}_{\mathbf{R}_{\mathfrak{v}}}} \cap Dom(\mathfrak{v}))} P_{\mathfrak{M}}(\mathfrak{v}(s))$$

Pero, como $Img_{\mathscr{M}_{\mathbf{R}_\mathfrak{v}}}$ está incluída en la imagen de $\mathscr{M}$,

$$\mathfrak{v}(Img_{\mathscr{M}_{\mathbf{R}_\mathfrak{v}}} \cap Dom(\mathfrak{v})) \subseteq Img(\mathfrak{M})$$

Entonces, y ya que por la forma en que armamos la lista de requerimientos por cada o distinto en $Img(\mathfrak{M})$ sólo hay un s en $Img_{\mathscr{M}_{\mathbf{R}_\mathfrak{v}}} \cap Dom(\mathfrak{v})$,

$$\sum_{\langle s,n\rangle \in \mathbf{T}} 2^{-n} \leq \sum_{s \in (Img(\mathcal{M}) - Dom(\mathfrak{v}))} P_{\mathscr{M}}(s) + \sum_{o \in Img(\mathfrak{M})} P_{\mathfrak{M}}(o)$$

Finalmente, para completar la demostración, utilizaremos la siguiente extensión trivial del lema 16.

Lema 24 *Sea $\mathscr{M}$ una máquina de cómputo finito.*

$$\sum_{s \notin Dom(\mathfrak{v})} P_{\mathscr{M}}(s) + \sum_{s \in Dom(\mathfrak{v})} P_{\mathfrak{M}}(\mathfrak{v}(s)) \leq 1$$

¿Y todo esto para qué?

Ahora que sabemos que $\mathbf{R}_\mathfrak{v}$ es un conjunto consistente de requerimientos, que puede ser construido por una máquina de cómputos finitos, veamos que la computadora que ellos determinan es la que necesitábamos.

Para ello nos basta con notar que, si $s \in Dom(\mathfrak{v})$

$$\begin{array}{lll} \langle s, n+1\rangle \in \mathbf{R}_\mathfrak{v} & \text{sii} & 2^{-n} < P_{\mathfrak{M}}(\mathfrak{v}(s)) \text{ y } \Pi_1^{\mathfrak{v}(s)}(\mathbf{R}_\mathfrak{v}) = \{s\} \\ & \text{sii} & n > H_{\mathfrak{M}}(\mathfrak{v}(s)) \text{ y } \Pi_1^{\mathfrak{v}(s)}(\mathbf{R}_\mathfrak{v}) = \{s\} \\ & \text{sii} & n \geq \lceil H_{\mathfrak{M}}(\mathfrak{v}(s))\rceil \text{ y } \Pi_1^{\mathfrak{v}(s)}(\mathbf{R}_\mathfrak{v}) = \{s\} \end{array}$$

Lo que nos lleva trivialmente a comprobar que la máquina determinada por $\mathbf{R}_\mathfrak{v}$ cumple con (3.1). *Q.E.D.*

Parte II

Cómputos eternos

Capítulo 4

Notas previas y definiciones

El otro tipo de máquina con el que trabajaremos es aquel en que no existe restricción en la cantidad de tiempo que tarde el proceso de cómputo. Esto no sólo implica que puede ser arbitrariamente grande, es decir, tardar tanto cuanto quiera, sino que también puede no detenerse nunca. De hecho, si una máquina de este tipo se detiene, entenderemos que falló el proceso de cómputo y no emitió resultado alguno.

4.1. Máquinas sobre cadenas

Estas máquinas sólo diferirán de las anteriores en el tiempo que le permitiremos tomar en computar. Como aquellas, estas máquinas leen de su cinta de entrada de izquierda a derecha (sin retroceder) y no pueden borrar su cinta de resultados.

Contrariamente a lo que definíamos en la sección anterior, para que un proceso de cómputo sea considerado una *computación exitosa* no es necesario que éste se detenga.

Definición 15 (Programa) *Igual que antes, llamaremos* programa *a la secuencia de dígitos (posiblemente infinita, ya que estas máquinas sí* pueden *leer todo el contenido de su cinta fuente, aunque no tienen que hacerlo necesariamente) que formada por los dígitos de la cinta fuente que lee la computadora durante un proceso de computación exitosa.*

Si una secuencia $\mathfrak{p}$ provoca una *computación exitosa* sobre la computadora $\mathscr{M}$ al ser escrita en su cinta de entrada, notaremos al resultado que se obtiene de ese proceso como $\mathscr{M}(\mathfrak{p})$. En este sentido, nombraremos p al programa correspondiente a la secuencia $\mathfrak{p}$, es decir, al prefijo de $\mathfrak{p}$ (posiblemente toda ella) que lea $\mathscr{M}^\infty$ durante ese proceso de cómputo exitoso.

Definición 16 (*Chaitin* $\triangleright$ Máquinas eternas para cadenas) *A las computadoras descriptas anteriormente, las llamaremos* máquinas de cómputos eternos (o infinitos) para cadenas.

Usualmente las nombraremos con letras cursivas mayúsculas, a las que le agregaremos un símbolo de infinito (como por ejemplo $\mathscr{M}^\infty$) para diferenciarlas de las máquinas de cómputo finito.

Así como podíamos pensar a las máquinas de cómputos finitos $\mathscr{M}$ como funciones $\mathscr{M} : 2^\star \to 2^\star$, podemos pensar a las máquinas de cómputos eternos $\mathscr{M}^\infty$ como funciones $\mathscr{M}^\infty : 2^{\leq\omega} \to 2^{\leq\omega}$, es decir, estas máquinas pueden leer y escribir secuencias tanto finitas como infinitas.

4.2. Máquinas sobre conjuntos

Similarmente a lo que vimos antes, tendremos un tipo de máquina de cómputo infinito que emite secuencias y otro que emite conjuntos de naturales vía una función de interpretación $\mathfrak{v}$.

Gracias a su capacidad de computar para siempre, estas máquinas pueden escribir infinitamente, entonces deberemos utilizar una función de representación que contemple esta particularidad.

Definición 17 (*Chaitin* ▷ Representación de conjuntos en secuencias) *Para representar conjuntos de naturales con secuencias (finitas o infinitas) de ceros y unos utilizaremos la función $\mathfrak{v} : 2^{\leq\omega} \to \mathscr{P}(\mathbb{N}_0)$ tal que, para todo $\mathfrak{s} \in 2^{\leq\omega}$ y para todo natural a, $a \in \mathfrak{v}(\mathfrak{s})$ sii hay a ceros entre dos unos en $\mathfrak{s}$.*

Definición 18 (*Chaitin* ▷ Máquinas eternas para conjuntos) *Cuando interpretemos el resultado de las máquinas de cómputo infinito como un conjunto (utilizando la función de representación $\mathfrak{v} : 2^{\leq\omega} \to \mathscr{P}(\mathbb{N}_0)$ vista más arriba), las llamaremos* máquinas de cómputo infinito para conjuntos.
Usualmente las nombraremos con letras góticas mayúsculas junto con un símbolo de infinito (como por ejemplo $\mathfrak{M}^\infty$).

Escribiremos $\mathfrak{M}^\infty(p) = A$ (respectivamente $\mathscr{M}^\infty(p) = s$) para indicar que la computadora $\mathfrak{M}^\infty$ (resp. $\mathscr{M}^\infty$) escribe como resultado una representación del conjunto de naturales A (resp. una secuencia s) en su cinta destino, si su cinta de entrada contiene al programa p.

Del mismo modo que en el caso de cómputos finitos, cada máquina sobre conjuntos puede verse como la función resultante de componer la función de representación con una máquina sobre secuencias.

$$\mathfrak{M}^\infty = \mathfrak{v} \circ \mathscr{M}^\infty$$

Otra vez, nos tomaremos la licencia de decir que una máquina $\mathfrak{M}^\infty$ imprime un conjunto si su máquina subyacente $\mathscr{M}^\infty$ imprime alguna de sus representaciones (vía $\mathfrak{v}$).

También impondremos sobre estas máquinas la restricción de que su conjunto de programas sea libre de prefijos.

4.3. Máquinas universales

Como declaramos antes, cuando no nos interese el tipo de resultado de estas máquinas, las nombraremos con letras mayúsculas de imprenta junto con el símbolo de infinito (como por ejemplo M^∞).

Definición 19 (Máquina Universal) *Una* máquina universal de cómputo infinito *es una máquina de cómputo infinito U^∞ tal que para cada máquina de cómputo infinito M^∞ existe una constante c_{M^∞} que cumple que: si $M^\infty(\mathfrak{p})$ está definida, hay un programa $\mathfrak{q}$ tal que $U^\infty(\mathfrak{q}) = M^\infty(\mathfrak{p})$ y $|\mathfrak{q}| \leq |\mathfrak{p}| + c_{M^\infty}$.*

Notemos que la demostración del teorema 4 puede replicarse sin modificaciones para el caso de cómputos eternos, ya que la posibilidad de enumerar recursivamente estas máquinas es consecuencia directa de la existencia de una enumeración recursiva de todas las tablas de instrucciones.

Teorema 25 (Chaitin) *Existe una máquina universal de cómputos eternos.*

En adelante, a menos que se indique expresamente lo contrario, cuando hablemos de máquinas universales de cómputos eternos nos estaremos refiriendo a la máquina que reconoce qué máquina particular deben imitar inspeccionando los prefijos de forma 0^i1 de sus programas. Es decir, cuando esta máquina reciba en su cinta de entrada una secuencia de forma 0^i1p, el cómputo que ésta provoque será igual al que provoca la secuencia p en la i-ésima máquina de la enumeración que la computadora universal maneja.

4.4. Definiciones fundamentales

En esta sección presentaremos definiciones establecidas por Chaitin en [2] para máquinas sobre conjuntos y que particularizaremos para el caso de los cómputos sobre palabras.

Programa minimal

Definición 20 (*Chaitin* ▷ Programa minimal) *Entre todos los programas que provocan que una máquina de cómputo infinito M^∞ imprima una secuencia finita o infinita (respectivamente un conjunto) o, llamaremos* programas minimales para *o* en *M^∞ a aquellos de los cuales M^∞ lee menos dígitos para completar su cómputo. Los notaremos $o^\star_{M^\infty}$.*

Información algorítmica

Definición 21 (*Chaitin* ▷ Información algorítmica) *Llamaremos* información algorítmica de *o* en *M^∞ a la cantidad de dígitos que M^∞ lee de los programas minimales para o en M^∞. La notaremos $I_{M^\infty}(o)$.*

Cuando nos estemos refiriendo a propiedades sobre computadores universales, escribiremos simplemente $I^\infty(o)$.

Probabilidad algorítmica

Definición 22 (Probabilidad algorítmica)

$$P_{M^\infty}(o) = \sum_{M^\infty(\mathfrak{p})=o \ y \ |\mathfrak{p}|<\infty} 2^{-|\mathfrak{p}|}$$

También utilizaremos $P^\infty(o)$ para los casos en los que intervengan máquinas universales.

Entropía algorítmica

Definición 23 (*Chaitin* ▷ Entropía algorítmica)

$$H_{M^\infty}(o) = -\log P_{M^\infty}(o)$$

$H^\infty(o)$ nos bastará para los casos en los que intervengan máquinas universales.

Los siguientes resultados se verifican de manera análoga al caso finito que vimos en capítulos anteriores.

Cota superior para la entropía

Proposición 26 (Chaitin) *Para toda secuencia finita o infinita (respectivamente conjunto) o, vale que*

$$H^\infty(o) \leq I^\infty(o)$$

Optimalidad de las máquinas universales

Teorema 27 (*Chaitin* ▷ Teorema de invarianza) *Para toda computadora de cómputos infinitos M^∞ existe un natural c_{M^∞} tal que, para toda secuencia finita o infinita (respectivamente conjunto) o, vale que*

1. $I_{U^\infty}(o) \leq I_{M^\infty}(o) + c_{M^\infty}$
2. $P_{M^\infty}(o) \cdot 2^{-c_{M^\infty}} \leq P_{U^\infty}(o)$
3. $H_{U^\infty}(o) \leq H_{M^\infty}(o) + c_{M^\infty}$

Capítulo 5

Teorema de la codificación

Un programa para un objeto debe poseer la información suficiente para identificar dicho objeto del resto de los objetos que comparten una clase con él. Así, una computadora podrá utilizar esta información para imprimir este elemento y no otro.

En ese sentido, podemos pensar a los programas como buscadores de objetos dentro de una clase; como un investigador privado buscando a una persona en una ciudad. Siguiendo con este juego, el largo de un programa puede verse como la cantidad de dinero que deberíamos pagarle al investigador para que encuentre a la persona. Así mismo, cuánto más chica sea la ciudad, o más fáciles de seguir sean los pasos del buscado, menos trabajo tendrá el investigador en cumplir su tarea y por lo tanto (asumiendo que tratamos con detectives justos) menos será lo que nos cobre.

Del mismo modo, podríamos preguntarnos si las cualidades de los elementos de una clase particular tienen alguna incidencia en la longitud de los programas mínimos para ellos. Por ejemplo: si fijamos de antemano la cantidad de elementos de los conjuntos que nos interesan, ¿podremos utilizar esta información para armarnos un programa que sea más corto que si no contásemos tal información? En este capítulo veremos que la respuesta a tal pregunta es afirmativa mostrando una clase de conjuntos para los cuales se materializa esta diferencia. Antes de eso, repasaremos algunas ideas que nos ayudarán a vislumbrar dónde pueden hallarse estas discrepancias.

Luego buscaremos cotas superiores para el tamaño de programas mínimos para ciertas familias de conjuntos.

En nuestro camino hallaremos familias para las cuales estas cotas son lo suficientemente ajustadas como para poder determinar que el teorema de la codificación se verifica para sus miembros.

También veremos otras familias para las cuales demostraremos que no puede existir una cota lo suficientemente ajustada como para permitirnos semejante declaración.

5.1. Nombres

"Pero, ¿cuál era su nombre?¿Cuál era el nombre de la rosa?"
UMBERTO ECO en *El nombre de la rosa*.

Al discutir la longitud de programas mínimos para ciertas clases de conjuntos, Chaitin [2] nos habla de *especificaciones* de naturales. Notemos que nos habla de *especificar* porque podemos pensar al proceso de imprimir cierto conjunto como una forma de especificar un natural. En este sentido, podemos ver a los programas como *especificaciones*, o *nombres* si se quiere, de los naturales.

Por ejemplo, una manera de especificar el natural n es imprimir el unitario que sólo contiene a aquel número (ie, $\{n\}$).

Una opción es siempre la descripción explícita: en la cinta de entrada escribimos una representación de n y dejamos que la máquina la copie en la cinta destino.

Para aplicar esta estrategia en nuestras máquinas que imprimen conjuntos, deberíamos utilizar la representación de n que impone nuestra función de representación $\mathfrak{v}$, es decir, su nombre en notación unaria: 10^n1.

Como vemos, este nombre para n ocupa $n+2$ *bits*; podemos intuir que debe haber otros nombres más cortos para n porque nuestra máquina (casi) no hace ningún esfuerzo durante el proceso que propone esta estrategia.

Podemos hacerla sudar un poco y darle una representación más económica de n, y que ella se ocupe de traducirla al formato que le impusimos al definirla.

Definición 24 (Representación autodelimitante) *Una forma conocida de especificar un natural a es utilizando su* representación autodelimitante*, que notaremos $\sigma(a)$. Esta representación consta de la longitud de la representación binaria del número seguida por dicha representación del número.*

Para saber donde termina la longitud y comienza el número, escribiremos la longitud intercalando ceros en las posiciones impares de la cinta. El primer uno que encontremos (que encuentre la máquina al leer su cinta fuente) en esas posiciones determinará el final de la longitud y el comienzo del número, del que ya conocemos la longitud de su representación.

Esta nombre del natural a, ocupa $\log(a) + 2 \cdot \log(\log(a))$ *bits*. El primer término es lo que cuesta la representación binaria de a, y el segundo, lo que cuesta la representación binaria de la longitud de la representación binaria de a con la modificación mencionada.

Proposición 28 (Chaitin) *Para todo $n \in \mathbb{N}$*

$$I^{\infty}(\{n\}) \leq \log n + 2 \cdot \log\log n + c_{\sigma}$$

Este tipo de especificación está íntimamente relacionado con un método directo de calcular n, pues ése es el único número (más allá de repeticiones del mismo número) que nuestra máquina podrá imprimir en su cinta destino, ya que imprimirá un unitario.

En otras palabras, este cálculo encierra un cómputo finito, ya que en un cierto tiempo, nuestra máquina deberá haber sido capaz de calcular el número que deseamos e imprimir su representación en la cinta destino.

Como vemos, al utilizar estos nombres no estamos aprovechando la capacidad de computar eternamente que tienen las máquinas que estudiamos en esta parte del trabajo. Para tratar de hacerlo deberíamos buscar especificaciones que puedan ser rectificadas a medida que avanzamos en el proceso de cómputo.

Por ejemplo, podríamos entender que un nombre para n es un programa que emita un conjunto de n elementos. Esta manera de especificar n es más indirecta que la anterior ya que nos estamos acercando asintóticamente a n, de manera que no existe un tiempo determinado en el cual podamos estar seguros de qué número representa nuestro programa, pues en algún momento posterior podría emitir un nuevo elemento.

Sólo podremos saber qué numero está representando el programa en el fin de los tiempos, pero si nuestro programa emite un conjunto finito, estará representando un número en particular, más allá de que nosotros, por nuestro caracter de mortales, no lleguemos nunca a saber cuál es.

Observación. Al tomar un programa para cualquier conjunto de n elementos como representación de un número n, estamos permitiendo que más de un conjunto sea nombre (intermedio) válido para un mismo número.

Esta falta de inyectividad sólo entorpecería nuestros próximos pasos teniendo que preocuparnos por detalles que no son relevantes a esta altura. Es por ello que en las próximas páginas tomaremos un conjunto en particular de n elementos como nombre de n: aquel que tiene los naturales $\{0; \ldots; n-1\}$ y nos referiremos a él como el *consecutivo* n.

Este nuevo nombre no puede ser más largo que el anterior. Veamos por qué: es muy fácil hallar una máquina que dado un programa que imprima un unitario simule su ejecución hasta obtener el único número que contiene ese conjunto y luego imprima todos los naturales menores a él.

Entonces, imprimir un consecutivo no puede ser más caro que imprimir su correspondiente unitario.

Proposición 29 (Chaitin) *Para todo $n \in \mathbb{N}$,*

$$I^{\infty}(\{0; \ldots; n-1\}) \leq I^{\infty}(\{n\}) + O(1)$$

Observación. En relación a lo que notábamos más arriba, fijémonos que la cualidad de acercamiento asintótico que mencionábamos es intrínseca al conjunto consecutivo, y es una característica que una máquina de cómputo finito no es capaz de aprovechar, porque nunca puede "saber" cuando se ha concluído de imprimir el conjunto. Si lo pudiera hacer la proposición 29 enunciaría una igualdad, pero la imposibilidad que comentábamos nos lleva a pensar que la otra desigualdad no debería existir. De hecho, en pocas páginas más adelante veremos que esta intuición es correcta.

Una manera bien conocida de "acortar programas", es decir, que nuestra máquina necesite menos datos para realizar sus tareas es darle mayor poder de cómputo brindándole la posibilidad de consultar un *oráculo*.

Definición 25 (Máquina con oráculo) *Ampliaremos el poder de cómputo de las máquinas que venimos utilizando permitiendo que consulten un* oráculo *capaz de responder fehacientemente preguntas de detención.*

En otras palabras, las máquinas que cuenten con este dispositivo podrán saber si cierto proceso de cómputo finito estará definido o no.

Agregaremos un apóstrofe a las notaciones referentes a este tipo de máquinas (por ejemplo: $\mathscr{U}'$, $\mathfrak{U}^{\infty\prime}$, I', H',$I^{\infty\prime}$, $H^{\infty\prime}$, etc.)

Sea una máquina como la que se muestra en el esquema de la figura 5.1.

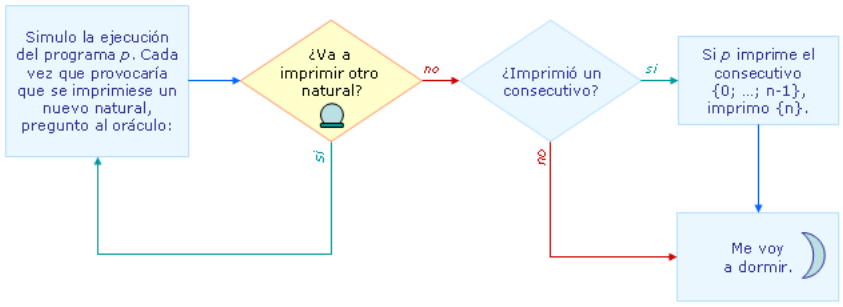

Figura 5.1: Descripción de la máquina con oráculo que nos permite hallar un consecutivo dado el unitario correspondiente.

El programa mínimo para un unitario sobre cómputos con oráculo no puede ser mayor, entonces, que el programa mínimo para un consecutivo (sobre cómputos sin oráculo) más alguna constante.

Proposición 30 (Chaitin) *Para todo $n \in \mathbb{N}$,*

$$I^{\infty\prime}(\{n\}) \leq I^{\infty}(\{0;\ldots;n-1\}) + O(1)$$

Así, imprimir un consecutivo sin utilizar el oráculo es un método más directo (en cuanto necesita menos trabajo de la máquina) y más costoso (en cantidad de *bits*) que imprimir el unitario para el mismo natural utilizando el oráculo.

Similarmente a lo que hicimos para verificar la proposición 29, si permitimos que nuestra máquina consulte un oráculo al realizar sus cómputos, imprimir el unitario $\{n\}$ seguirá siendo una manera más directa de especificar n que imprimir el consecutivo $\{0;\ldots;n-1\}$.

Proposición 31 (Chaitin)

$$I^{\infty\prime}(\{0;\ldots;n-1\}) \leq I^{\infty\prime}(\{n\}) + O(1)$$

5.1.1. Descubriendo diferencias

Si bien hemos podido escalonar los nombres para n de manera de saber cuál no puede ser más grande que cuál, en esta sección descubriremos que hay números para los cuales unos nombres son realmente más cortos que otros. Para ello nos dedicaremos a demostrar el siguiente teorema.

Teorema 32 (Chaitin)

1. *Dado k, el menor n tal que $n \geq 2^k$ y $I^\infty(\{n\}) \geq k$, cumple que*

   a) $n < 2^{k+1}$

   b) $I^\infty(\{n\}) = k + O(\log k)$

   c) $I^\infty(\{0; \ldots; n-1\}) \leq \log k + O(\log \log k)$

2. *Dado k, el menor n tal que $n \geq 2^k$ y $I^\infty(\{0; \ldots; n-1\}) \geq k$, cumple que*

   a) $n < 2^{k+1}$

   b) $I^\infty(\{0; \ldots; n-1\}) = k + O(\log k)$

   c) $I^{\infty\prime}(\{n\}) \leq \log k + O(\log \log k)$

3. *Dado k, el menor n tal que $n \geq 2^k$ y $I^{\infty\prime}(\{n\}) \geq k$, cumple que*

   a) $n < 2^{k+1}$

   b) $I^{\infty\prime}(\{n\}) = k + O(\log k)$

   c) $I^{\infty\prime}(\{0; \ldots; n-1\}) \leq \log k + O(\log \log k)$

Como podemos apreciar, este teorema nos muestra que, para los números por él considerados, se hacen estrictas las desigualdades que vimos en la sección anterior.

Parte *a*

Tomemos todos los programas de tamaño a lo sumo $k-1$. Si con cada uno de ellos pudiéramos especificar (de la manera que nos interesa) un natural distinto, podríamos especificar, a lo sumo, 2^{k-1} naturales (ya que esa es la cardinalidad del conjunto libre de prefijos más grande formado por palabras de tamaño, a lo sumo, $k-1$).

Aunque todos estos programas se utilizaran para especificar naturales en el segmento $[2^k; 2^{k+1})$ sólo nos alcanzaría para la mitad de dichos naturales. Por lo tanto, hay por lo menos 2^{k-1} naturales en el segmento $[2^k; 2^{k+1})$ que son más caros de especificar que k, es decir, más caros que (la parte entera por abajo) de su logaritmo.

Parte *b*

Sea cual fuere el tipo de especificación que escojamos, hemos elegido nuestros n de manera tal que el precio de esa especificación no es menor que $\lfloor \log n \rfloor$; pero, ¿cuánto más cara podrá ser?

Hemos seleccionado a cada n_k de modo que sea mayor o igual a 2^k. Entonces, podemos escribir a n_k como la suma entre 2^k y un resto no negativo r_k. Aprovechando esta cualidad de n_k podríamos utilizar una máquina $\mathfrak{M}^\infty$ tal que, dada una codificación de k seguida por una codificación de r_k, efectúe la suma que mencionábamos anteriormente y consiga así el valor n_k, de forma tal de poder imprimir la especificación que deseemos. Luego, la especificación de n_k en esta máquina no será más cara que lo que nos cueste codificar k y r_k.

Para ello, podemos utilizar la representación auto-delimitante que ya utilizamos para la cota superior de estos nombres (proposición 28). Utilizando esta estrategia podríamos codificar k utilizando $\log(k) + 2 \cdot \log(\log(k))$ *bits*. Además, como ya hemos visto, n_k es menor que 2^{k+1}, lo que nos permite inferir que r_k es menor que 2^k. Entonces, su representación auto-delimitante no será mayor que $k + 2 \cdot \log(k)$.

Así, hallamos que la especificación de n_k en $\mathfrak{M}^\infty$, y por lo tanto en cualquier máquina universal, no es mayor que $k + O(\log(k))$; o, dicho de otro modo, $\lfloor \log n \rfloor + O(\log(\lfloor \log n \rfloor))$.

Parte *c*

Hablábamos anteriormente de métodos más indirectos de especificación. Estos métodos podrían tener la ventaja de ser más económicos que los directos, ya que aprovechan características que las especificaciones directas no utilizan. Esta vez estudiaremos las diferencias caso por caso.

Unitarios vs. consecutivos

Si tomamos como la especificación directa de n al programa que imprime su unitario correspondiente, veremos a continuación que, para los números que escogimos como candidatos, el programa que imprime su consecutivo no es más caro que $\log(k) + O(\log(\log(k)))$.

Sea $\mathfrak{M}^\infty_{1^\circ}$ una máquina de cómputos eternos que simula el comportamiento de la máquina universal $\mathfrak{U}^\infty$ hasta que dicha máquina imprima el primer natural, en ese momento imprime el mencionado número y se va a "dormir" (es decir, deja de leer su cinta de entrada), provocando como resultado de su proceso de cómputo al unitario correspondiente al natural que imprimiría $\mathfrak{U}^\infty$.

Gracias al teorema de invarianza podemos deducir que existe un natural $c_{\mathfrak{M}^\infty_{1^\circ}}$ tal que, para todo $a \in \mathbb{N}$,

$$I^\infty(\{a\}) \leq I^\infty_{\mathfrak{M}^\infty_{1^\circ}}(\{a\}) + c_{\mathfrak{M}^\infty_{1^\circ}}$$

Sea, ahora, $\mathfrak{M}^\infty$ una máquina de cómputo infinito tal que, al recibir cierto valor k en su cinta de entrada, busca el menor $n \geq 2^k$ tal que $I^\infty(\{n\}) \geq k$.

Para ello, tomará como primer candidato c a 2^k e imprimirá todos los naturales menores a él, imprimiendo así el consecutivo $\{0; ...; c-1\}$. Luego de hacerlo, $\mathfrak{M}^\infty$ simulará la máquina universal $\mathfrak{U}^\infty$ realizando un proceso de *dovetailing* sobre los programas de longitud menor a $k - c_{\mathfrak{M}^\infty_{1\mathrm{O}}}$.

Si en este proceso encuentra un programa que imprima el unitario $\{c\}$, este programa provocaría en la máquina $\mathfrak{M}^\infty_{1\mathrm{O}}$ un proceso que resultaría en ese unitario. Como, además, el programa en cuestión no mide más de $k - c_{\mathfrak{M}^\infty_{1\mathrm{O}}} - 1$ dígitos de largo,

$$I^\infty_{\mathfrak{M}^\infty_{1\mathrm{O}}}(\{c\}) \leq k - c_{\mathfrak{M}^\infty_{1\mathrm{O}}} - 1$$

Entonces, por lo que habíamos visto más arriba,

$$I^\infty(\{a\}) < k$$

Esto quiere decir que nuestro candidato c no es el natural n que estábamos buscando (ya que no cumple que el programa mínimo para el unitario n es más caro que k). En ese caso, $\mathfrak{M}^\infty$ tomará como candidato al siguiente número $(c+1)$ y repetirá el proceso.

Notemos que la máquina nunca sabe cuándo ha impreso el candidato c que cumple que $I^\infty(\{c\}) \geq k$, pero una vez que lo ha impreso es seguro que nunca más imprimirá nuevos números en su cinta de salida.

Esta aproximación a n es más indirecta respecto de imprimir el unitario n; como puede verse en el proceso descripto más arriba, la aproximación al valor final es asintótica.

Así, podemos ver que simplemente brindando k podemos obtener el menor $n \geq 2^k$ tal que $I^\infty(\{n\}) \geq k$; y, para brindar k sólo necesitamos tantos *bits* como el logaritmo de k más un término del orden del doble logaritmo de k.

De esta manera hemos llegado a demostrar el teorema 32.1, descubriendo la primera de las brechas que estuvimos buscando: el nombre en consecutivos para este tipo de naturales es necesariamente más pequeño que el nombre en unitarios.

Notemos que en la discusión precendente nunca nos fijamos si durante el proceso de cómputo nuestra máquina consulta a un oráculo para cumplir sus tareas. Luego, de la misma manera podemos obtener los resultados análogos para cómputos con oráculo que se enuncian en el teorema 32.3.

Consecutivos vs. unitarios

Sea, ahora, una máquina $\mathfrak{M}^{\infty\prime}$ de cómputo infinito con oráculo tal que al recibir cierto valor k en su cinta de entrada, busca el menor $n \geq 2^k$ tal que $I^\infty(\{0; \ldots; n-1\}) \geq k$, e imprime el unitario $\{n\}$.

Para ello, realizará el proceso que muestra el esquema de la figura 5.2, en el cual, con el objetivo de determinar cuáles programas de longitud menor a k imprimen consecutivos y cuáles no lo hacen, vamos a quedarnos con los programas que nos interesa y descartar al resto. Finalmente sólo debemos calcular el n que deseamos basándonos en los resultados de los programas seleccionados.

Así se completa la demostración del del teorema 32. *Q.E.D.*

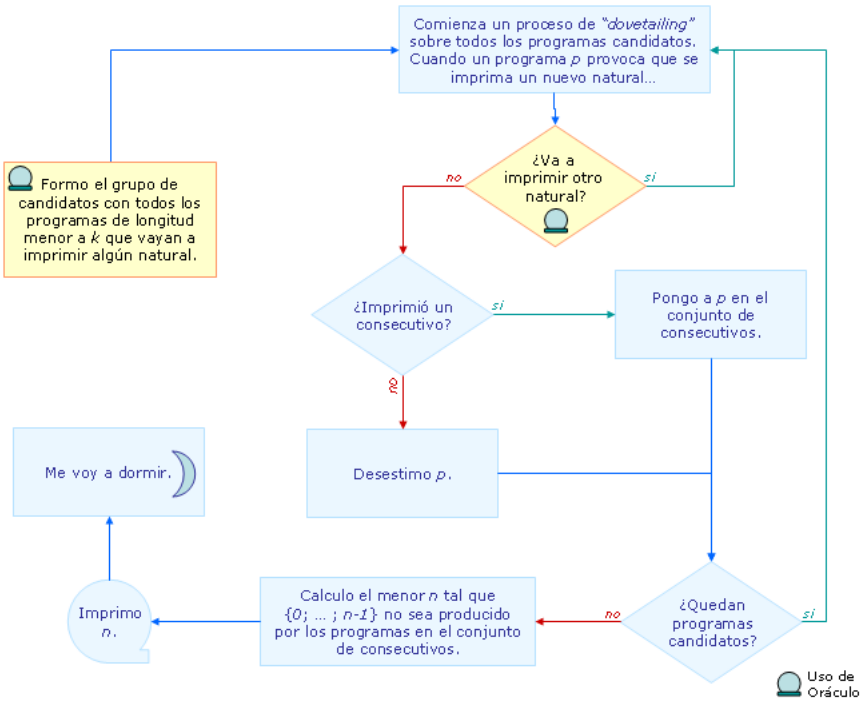

Figura 5.2: Descripción de la máquina que nos permite hallar un unitario dado el consecutivo correspondiente.

5.2. Familias uniformes

Las primeras familias que visitaremos en este viaje son aquellas que nos deparan pocas sorpresas. Aquellas cuyos miembros tienen una particularidad común que los identifica tan fuertemente que vamos a poder encontrar una cota lo suficientemente ajustada para la longitud de sus programas mínimos como para poder verificar el teorema de la codificación.

Veamos a qué familias nos estamos refiriendo.

Definición 26 (Familias Δ) *Llamaremos Δ_i a la familia de conjuntos de cardinal i. Así mismo, llamaremos A_i a los miembros de la familia Δ_i.*

Demostraremos en esta sección el siguiente teorema.

Teorema 33 *Dada una familia uniforme Δ_i, existe un natural c_{Δ_i} tal que, para todo conjunto de naturales $A_i \in \Delta_i$,*

$$H^{\infty}(A_i) \leq I^{\infty}(A_i) \leq H^{\infty}(A_i) + c_{\Delta_i}$$

Para ello, utilizaremos la cota para la entropía $H^\infty(A) \leq I^\infty(A)$ que vimos anteriormente (propiedad 26) y el siguiente teorema que nos dará una cota superior para el tamaño de los programas mínimos de las familias uniformes. Para su demostración utilizaremos las definiciones establecidas a continuación.

Definición 27 (Tupla) *Tomemos una biyección $\varphi : 2^\star \times 2^\star \to 2^\star$ y definamos: $\langle s \rangle$ como la misma cadena s y, para $n \geq 2$, $\langle s_1, \ldots, s_n \rangle$ como la cadena $\varphi(\langle s_1, \ldots, s_{n-1} \rangle, s_n)$.*

Definición 28 (Conjunto subyacente) *Si t es una tupla $\langle s_1, \ldots, s_n \rangle$, entonces llamaremos* conjunto subyacente *de t (notado $\sim t$) al conjunto de los elementos de t, i.e. $\sim t = \{s_1, \ldots, s_n\}$.*

Definición 29 (Tupla canónica) *Dado un conjunto de naturales A, llamaremos* tupla canónica *de A (notada $\langle A \rangle$) a la tupla formada por los elementos de A (sin repeticiones) dispuestos en orden creciente.*

Teorema 34 *Dada una familia uniforme Δ_i, existe un natural c_{Δ_i} tal que, para todo conjunto de naturales $A_i \in \Delta_i$,*

$$I^\infty(A_i) \leq H^\infty(A_i) + c_{\Delta_i}$$

Demostración. Sea $\mathscr{M}_i$ una computadora de cómputos finitos que, al recibir al programa p en su cinta de entrada, simula a una máquina universal de cómputos eternos $\mathfrak{U}^\infty$ cuya cinta de entrada comienza con p.

Cuando $\mathfrak{U}^\infty$ escribe i elementos distintos, si leyó exactamente a p (es decir, ni un dígito mas ni uno menos), $\mathscr{M}_i$ imprime la tupla canónica formada por esos i elementos y termina; en otro caso, se indefine.

Luego, por cada programa p tal que $\mathfrak{U}^\infty(p) = A_i$, existe un prefijo q de p tal que $\mathscr{M}_i(q) = \langle A_i \rangle$.

Notemos, sin embargo, que también existen programas q tales que si bien $\mathscr{M}_i(q) = \langle A_i \rangle$, ninguna de sus extensiones provocan que tal máquina imprima miembros de Δ_i.

Entonces

$$P^\infty(A_i) \leq P_{\mathscr{M}_i}(\langle A_i \rangle)$$

Además, por el teorema de invarianza (8), podemos afirmar que

$$P_{\mathscr{M}_i}(\langle A_i \rangle) \leq P(\langle A_i \rangle) \cdot 2^{c_{\mathscr{M}_i}}.$$

Luego, por el teorema 10, existe una constante m_i tal que

$$P^\infty(A_i) \leq P(\langle A_i \rangle) \cdot 2^{c_{\mathscr{M}_i}} \leq 2^{-I(\langle A_i \rangle) + m_i + c_{\mathscr{M}_i}}$$

Notemos, ahora, la siguiente propiedad.

Proposición 35 *Existe una constante $c_\sim$ tal que para toda tupla t*

$$I^\infty(\sim t) \leq I(t) + c_\sim$$

Demostración. En primer lugar, notemos que existe una máquina $\mathfrak{M}^{\infty}_{\sim}$ tal que para todo programa p simula una máquina universal $\mathscr{U}$ y si $\mathscr{U}(p) = \langle s_1, \ldots, s_n \rangle$ entonces $\mathfrak{M}^{\infty}_{\sim}(p) = \sim \langle s_1, \ldots, s_n \rangle$. Luego, el teorema de invarianza (teorema 27) nos lleva a comprobar que, para toda tupla t,

$$I^{\infty}(\sim t) \leq I(t) + c_{\mathfrak{M}^{\infty}_{\sim}}$$

que es el resultado que queríamos obtener (con $c_{\sim} = c_{\mathfrak{M}^{\infty}_{\sim}}$). *Q.E.D.*

Juntando esta Proposición con la inecuación anterior llegamos a

$$P^{\infty}(A_i) \leq 2^{-I^{\infty}(A_i) + m_i + c_{\mathscr{M}_i} + c_{\sim}}$$

y finalmente

$$I^{\infty}(A_k) \leq H^{\infty}(A_k) + m_i + c_{\mathscr{M}_i} + c_{\sim}$$

que es lo queríamos demostrar (con $c_{\Delta_i} = m_i + c_{\mathscr{M}_i} + c_{\sim}$). *Q.E.D.*

5.3. Familias consecutivas

Ampliemos, ahora, nuestros horizontes hacia familias más complejas.

Definición 30 (Familias consecutivas) *Sea $\mathcal{C}$ un conjunto recursivamente enumerable, y llamemos $\mathbf{P}_{\mathcal{C}}$ al procedimiento que lista sus elementos. Llamaremos $\Phi_{\mathbf{P}_{\mathcal{C}}}$ a la familia de conjuntos finitos definida de la siguiente manera:*

$A \in \Phi_{P_{\mathcal{C}}}$ *sii* *A está formado por los $|A|$ primeros elementos que lista $P_{\mathcal{C}}$*

Resulta claro que, por cada natural n, sólo hay un conjunto de cardinal n en Φ_P. Llamaremos, entonces, $\Phi_P[n]$ al conjunto en Φ_P que tiene n elementos.

5.3.1. Una cota superior para programas mínimos

Teorema 36 *Sea $\Phi_{\mathcal{P}}$ uuna familia consecutiva. Para todo $A \in \Phi_{\mathcal{P}}$,*

$$I^{\infty}(A) \leq H^{\infty}(A) + O(\log H^{\infty}(A))$$

Demostración. Como sólo hay un conjunto de cada cardinalidad en $\Phi_{\mathcal{P}}$, para armarnos un programa mínimo que imprima a un miembro de estas familias, nos basta con conocer la cantidad de elementos del conjunto en cuestión. Pero, ¿habrá alguna forma de deducir esta información de un conjunto en función de su entropía?

En principio, notemos que cada familiar de una familia Φ_P pertenece a una familia Δ distinta. Entonces, si pudiésemos encontrar una manera de indicar a qué familia Δ pertenece el conjunto que queremos imprimir, estaríamos identificando unívocamente a dicho conjunto.

Para ver cómo aprovechar esta particularidad veamos de qué manera se van formando los conjuntos de programas definidos a continuación.

Definición 31 *Llamemos $\mathcal{R}_n^t$ al conjunto de las cadenas que, escritas al principio de la cinta de entrada de una máquina universal $\mathfrak{U}^\infty$, provocan que dicha máquina imprima un conjunto perteneciente a la familia Δ_n si se ejecutan t pasos de computación.*
Es decir, cada palabra en $\mathcal{R}_n^t$ imprime un conjunto de exactamente n elementos en t pasos de cómputo de una máquina universal de cómputos eternos.

Hablamos de conjuntos de cadenas (y no de secuencias) pues en una computación de t pasos, $\mathfrak{U}^\infty$ puede leer a lo sumo t caracteres de su cinta de entrada.

Notemos que, si fijamos n, $\#\mathcal{R}_n^t$ va fluctuando a medida que aumenta t. Esta fluctuación se debe a que si en t pasos de cómputo un proceso imprimía un conjunto de n elementos, en el siguiente paso puede imprimir un elemento nuevo (con lo que migraría su aporte a $\mathcal{R}_{n+1}^{t+1}$) o se puede indefinir.

En cualquiera de estos casos, el programa quitaría su aporte de $\mathcal{R}_n^t$, pero esto no nos autoriza a pensar que $\#\mathcal{R}_n^t$ sólo decrece a medida que aumenta t, pues en ese mismo momento (mientras $\mathcal{R}_n^t$ pierde aportes por las causas menciondas) puede ganar nuevos aportes por la impresión de nuevos elementos por parte de programas en $\mathcal{R}_{n-1}^t$.

Por ello, no podemos saber si, fijando n, cada $\mathcal{R}_n^t$ será más grande o más pequeño en la próxima generación. Sin embargo, podemos aprovechar que cada programa de $\mathfrak{U}^\infty$ sólo puede migrar su aporte hacia conjuntos $\mathcal{R}_n^t$ con n más grande a medida que pasa el tiempo, para hallar conjuntos de cadenas que pueden armarse incrementalmente generación por generación.

Definición 32 (Familias incrementales) *Llamemos Γ_n a la familia de conjuntos tal que todos sus miembros tienen menos de n elementos.*

Obviamente, las familias incrementales no son más que agrupaciones de familias uniformes.

$$\Gamma_n = \bigcup_{i=0}^{n-1} \Delta_i$$

Como hicimos con las familias uniformes, veamos cómo se van conformando los conjuntos de cadenas que imprimen a los miembros de las familias incrementales.

Definición 33 *Llamemos $\mathcal{Q}_n^t$ al conjunto de las cadenas que, escritas al comienzo de la cinta fuente de una máquina universal $\mathfrak{U}^\infty$, provocan que dicha máquina imprima un conjunto perteneciente a la familia Γ_n si se ejecutan t pasos de computación.*
Es decir, cada palabra en $\mathcal{Q}_n^t$ imprime un conjunto de menos de n elementos en t pasos de cómputo de una máquina universal de cómputos eternos.

Observación. La relación que guardan las familias incrementales y las familias uniformes se conserva entre los conjuntos $\mathcal{R}^t$ y $\mathcal{Q}^t$.

$$\mathcal{Q}_n^t = \bigcup_{i=0}^{n-1} \mathcal{R}_i^t$$

La n-ésima carrera

Propongámonos un juego: que todas las secuencias binarias corran una carrera, "la n-ésima carrera". Esta carrera se corre sobre un camino de baldosas y, como en cualquier carrera, todas las corredoras deberán comenzar en la primer baldosa del camino. Pero deberán avanzar por turnos y siguiendo la siguiente regla: de las secuencias que están en la t-ésima baldosa sólo podrán avanzar a la siguiente casilla aquellas que, escritas en la cinta de entrada de $\mathfrak{U}^\infty$, provocan que dicha máquina imprima un conjunto de cardinal menor a n en $t+1$ pasos de cómputo.

Para referirnos a una baldosa, deberíamos utilizar un nombre, pero no gastemos demasiado tiempo en buscar uno: simplemente llamemos $\mathcal{Q}_n^i$ a la baldosa i.

Ahora bien, en el turno t, ¿cuántas secuencias habrá en cada baldosa? Para tener una idea de esta cantidad, llamemos $Q^t(n)$ a la probabilidad de que una secuencia formada por tiradas de monedas esté en la baldosa $\mathcal{Q}_n^t$; o de otro modo:

Definición 34 *Llamemos $Q^t(n)$ a la probabilidad de que una secuencia construida a partir de tiradas de monedas provoque que $\mathfrak{U}^\infty$ imprima un conjunto de menos de n elementos (ie, perteneciente a Γ_n) si se ejecutan t pasos de cómputo.*

Como ya habíamos dicho, todas las secuencias comenzarán en la casilla inicial. De hecho, si no permitimos ningún paso de cómputo, $\mathfrak{U}^\infty$ no puede leer siquiera un caracter de su cinta de entrada, menos aún escribir alguno. Lo que indica que todas las competidoras, al menos hasta el minuto cero, imprimen al conjunto vacío. Entonces:

$$\mathcal{Q}_n^0 = 2^\star \quad \Rightarrow \quad Q_n^0 = 1$$

A medida que avancen los turnos, cada secuencia tendrá la posibilidad de imprimir más naturales distintos, ya que la máquina tendrá cada vez más tiempo para realizar sus cómputos.

Aquellas secuencias que provoquen la impresión de un conjunto de cardinal mayor a n, o aquella que provoque que el cómputo se detenga, será descalificada inmediatamente, y no podrá avanzar a la siguiente casilla. Luego, la cantidad de secuencias sólo puede disminuir a medida que avanzan los turnos. Es decir:

$$Q_n^t \geq Q_n^{t+1} \tag{5.1}$$

Así, cómo habíamos establecido, sólo seguiran avanzando aquellas secuencias que no impriman más de n elementos, y serán sólo éstas las que seguirán corriendo indefinidamente.

Definición 35 *Llamemos $\mathcal{Q}_n$ al conjunto de las secuencias que, escritas en la cinta de entrada de una máquina universal $\mathfrak{U}^\infty$, provocan que dicha máquina imprima un conjunto de menos de n elementos (ie, perteneciente a la familia Γ_n).*

Como antes, definamos la noción de probabilidad asociada a $\mathcal{Q}_n$.

Definición 36 *Llamemos $Q^\infty(n)$ a la probabilidad de que una secuencia construida a partir de tiradas de monedas provoque que $\mathfrak{U}^\infty$ imprima un conjunto de menos de n elementos (ie, perteneciente a Γ_n).*

Ahora que podemos, pasemos en limpio lo que acabamos de observar.

$$\lim_{t\to\infty} Q_n^t = Q_n \tag{5.2}$$

Notemos que, para una carrera n, no es difícil calcular cuántas secuencias hay en cada baldosa t. Como $\mathfrak{U}^\infty$ sólo puede leer, a lo sumo, los t primeros dígitos de cada secuencia, y sólo puede ejecutar t pasos de computación, para calcular $\mathcal{Q}_n^t$ (o $Q^t(n)$) nos basta con simular los t primeros pasos de $\mathfrak{U}^\infty$ con cada uno de los 2^t comienzos de cinta posibles.

Sin embargo lo que nos interesaba realmente es identificar un n, entonces deberíamos ver qué relación cumplen estas cantidades en función de los naturales a los que se refieren.

En principio notemos que, dada la íntima relación que existe entre los conjuntos de secuencias $\mathcal{Q}_n$ y las familias incrementales Γ_n, resulta claro que la inclusión entre familias incrementales sucesivas provoca el mismo efecto sobre conjuntos $\mathcal{Q}$. Es decir:

$$\Gamma_n \subseteq \Gamma_{n+1} \Rightarrow \mathcal{Q}_n \subseteq \mathcal{Q}_{n+1}$$

Lo que, su vez, implica que

$$Q^\infty(n) \leq Q^\infty(n+1)$$

Como habíamos dicho, $Q^\infty(n)$ puede interpretarse como la probabilidad de imprimir un conjunto de menos de n elementos. Entonces, si A es un conjunto finito, vale que $Q^\infty(\#A) = \sum_{B|\#B<\#A} P^\infty(B)$. Luego,

$$\begin{array}{rcl} Q^\infty(\#A+1) - Q^\infty(\#A) & = & \sum_{B|\#B\leq\#A} P^\infty(B) - \sum_{B|\#B<\#A} P^\infty(B) \\ & = & \sum_{B|\#B=\#A} P^\infty(B) \\ & \geq & P^\infty(A) \end{array}$$

Lo que da fundamento al siguiente lema.

Lema 37 (Chaitin) *Sea A un conjunto finito de naturales. Si $P^\infty(A) > 2^{-n}$, existe un racional x (de forma $k/2^n$) en el intervalo $(Q^\infty(\#A); Q^\infty(\#A+1))$.*

Entonces, si conociéramos $Q^\infty(n)$ para cada n, sería muy fácil conseguir el n que buscamos, ya que simplemente deberíamos recorrer la sucesión $\{Q^\infty(n)\}_{n=0}^\infty$ comenzando desde el cero y esperando al valor de n que provoque que $Q^\infty(n) < x$.

Sin embargo, para una máquina sin oráculo no es sencillo conseguir los valores de $Q^\infty(n)$, ya que esto implicaría conocer los resultados la máquina universal para cada programa posible (lo que incluye al comportamiento de esta misma máquina).

Pero veamos cómo van formándose generacionalmente los conjuntos $\mathcal{Q}_n^t$ con respecto a n.

Que comience la "multicarrera"

¿Qué ocurriría si todas las n-ésimas carreras se corrieran simultáneamente, por andariveles paralelos, uno por cada n?

Al principio, en cada camino veríamos avanzar a todas las secuencias juntas. Cuando alguna de ellas fuera descalificada por detener el cómputo de $\mathfrak{U}^\infty$, desaparecería de todos los caminos. Sin embargo, si fuera descalificada de algún camino por haber provocado la impresión de un conjunto de cardinal mayor al número que identifica el andarivel correspondiente, dicha secuencia sólo desaparecería de ese camino, pero no de los caminos mayores (es decir, con n más grandes).

Esto se debe a que, si una secuencia imprime (o al menos está imprimendo en un momento t) un conjunto de menos de n elementos, también imprime

(está imprimiendo) un conjunto de menos de $n+1$ elementos, y de menos de $n+2$, y de $n+3$, etc. Luego

$$\mu\left(\mathcal{Q}_n^t 2^\omega\right) \leq \mu\left(\mathcal{Q}_{n+1}^t 2^\omega\right) \quad \text{y, equivalentemente,} \quad Q^t(n) \leq Q^t(n+1) \tag{5.3}$$

Bien, ahora deberíamos preguntarnos cómo podemos utilizar todo esto para identificar a una familia Γ_n particular.

Ocupen sus localidades

Imaginemos la sala de un teatro. Una sala grande. Lo suficientemente grande como para que haya una silla por cada racional de la forma $k/2^l$.

Nuestro teatro también tiene una sala de espera espaciosa. Tanto es así que están allí, esperando por entrar a la sala, una persona por cada natural. Como se trata de una velada de gala, la asistencia requiere rigurosa etiqueta. Aun cumpliendo con los formalismos de la ocasión, cada una de estas personas tiene (los caballeros en sus smokings y las damas en sus largos vestidos) una inscripción $Q^\infty(n)$ distinta (con $n = 0; 1; \ldots$).

Habíamos visto más arriba que, para calcular cada valor $Q^t(n)$, nos bastaba con ejecutar los t primeros pasos de cómputo sobre todas las posibles cadenas, que tienen tamaño t. De todas ellas, que son 2^t palabras, sólo una cantidad k imprimirán conjuntos de cardinal menor a n. Luego, $Q^t(n)$ será un racional de la forma $k/2^t$.

Aprovecharemos este hecho para ubicar a las personas en la sala. En cada minuto t, permitiremos a la persona $Q^\infty(n)$ que ocupe la localidad correspondiente a $Q^t(n)$. (Es posible que más de una persona le toque el mismo lugar en un minuto dado, pero no importa porque las sillas son lo suficientemente cómodas, lo que es suficientemente oportuno, por supuesto.)

Por (5.3), sabemos que en ningún minuto t ocurrirá que una persona $Q^\infty(m)$ se siente a la izquierda de otra con identificación $Q^\infty(n)$ si m es más grande que n.

A su vez, por lo visto en (5.1) sabemos que cada persona se irá acomodando en sillas a su izquierda a medida que avanzan los minutos.

Entonces, si sabemos que la persona que estamos buscando es aquella que tiene el mayor número de asiento menor que cierto racional r (de la forma $k/2^t$), nos basta con quedarnos en el escenario mirando esa silla y esperar a que las personas se acomoden.

Tal vez debamos esperar bastante. Más o menos toda la eternidad, ya que nunca podremos estar seguros que alguien a la derecha de la silla no vaya a abandonar su asiento y a ocupar uno a la izquierda (convirtiéndose asi en nuestro nuevo candidato, por lo comentado en los párrafos precedentes).

Pero aún así, podemos estar seguros que siempre nos iremos acercando desde "abajo", ya que si nuestro candidato actual es el de saco $Q^\infty(n)$ sólo podrá ser reemplazado por una persona con saco $Q^\infty(m)$ con m mayor a n.

De este modo, si nuestro candidato es el de saco $Q^\infty(n)$, la familia que estamos buscando es Γ_n. Luego, sólo nos resta hallar una manera económica de representar un racional r de la forma $k/2^t$.

Definición 37 (Representación asintótica de un racional) *Sea r un racional de la forma $j/2^n$. Llamaremos* representación asintótica de r *(notada $\alpha(r)$) al programa formado por la representación binaria de r apropiadamente mezclada con un programa mínimo para un conjunto de n elementos.*

Hay un grupo de máquinas $\mathfrak{M}_\alpha^\infty$ para las cuales es muy fácil, dada una representación como las que acabamos de definir, conseguir el valor codificado por ella.

Para eso, dado $\alpha(r)$, sólo deben simular la computadora universal $\mathfrak{U}^\infty$ con el programa que tienen escrito en su cinta de entrada y, cada vez que $\mathfrak{U}^\infty$ imprimiría un nuevo natural, la máquina leerá el siguiente *bit* de su cinta de entrada, que será el próximo dígito de la representación binaria de r.

Como r es de la forma $j/2^n$, la representación binaria de r sólo ocupa n *bits*. Entonces, por cada nuevo elemento del conjunto que imprima el programa inscripto en $\alpha(r)$, hay un dígito de la representación binaria de r.

Observación. No deberíamos seguir avanzando sin prestar atención a dos puntos. Primero, dada una representación asintótica $\alpha(r)$, estas máquinas $\mathfrak{M}_\alpha^\infty$ nunca saben cuándo han terminado de leer la representación de r. Es por ello que se puede pensar que, más que llegar a determinar quién es r, estas $\mathfrak{M}_\alpha^\infty$ hallan una sucesión creciente de racionales que converge a r. Este también es buen argumento para intuir que esta representación es más económica que la autodelimitante, ya que dada ésta última fácilmente obtenemos la anterior, pero no parece que ocurriera lo mismo en el caso contrario.

En segundo lugar, y relacionado con lo que acabamos de remarcar, armar una representación como estas no parece tarea trivial (para una máquina) pero, ¿qué importa? Es decir, armar la representación parece demandar algunas gotas de sudor, un precedente necesario para una representación económica en longitud.

Lema 38 *Sea $\Phi_\mathcal{P}$ una familia consecutiva y x un racional de la forma $j/2^n$. Existe una constante $c_{\Phi_\mathcal{P}}$ tal que*

$$I^\infty\left(\Phi_\mathcal{P}[\text{máx}\{m : Q^\infty(m) \leq x\}]\right) \leq I^\infty\left(\{0; \ldots; |x_{(2)}| - 1\}\right) + |x_{(2)}| + c_{\Phi_\mathcal{P}}$$

donde $x_{(2)}$ es la representación binaria de x (cuya longitud no se extiende más allá de los n dígitos).

Demostración. Sea ${\mathfrak{M}_\alpha^\infty}_\mathcal{P}$ una máquina que cumple que

$${\mathfrak{M}_\alpha^\infty}_\mathcal{P}(\alpha(x)) = \Phi_\mathcal{P}[\text{máx}\{m : Q^\infty(m) \leq x\}]$$

Es decir, esta máquina será la encargada de decirnos qué familiar de $\Phi_\mathcal{P}$ está sentado más cerca, por izquierda, a la butaca x.

Para ello, y si llamamos x^t al prefijo de tamaño t de la representación binaria de x, a medida que se llena la sala de nuestro teatro irá mirando, en cada minuto

t, la butaca x^t e irá arriesgando candidatos de acuerdo a lo que "ve" en ese minuto.

Nunca sabrá (y nunca sabremos) cuándo realmente halló al familiar indicado, pero como los valores x^t van creciendo a medida que se acercan a x (es decir, va revisando la sala de izquierda a derecha) y los valores de $Q^t(n)$ no pueden incrementarse a medida que avanza t (es decir, la sala se llena de derecha a izquierda) nunca va a presentarnos un candidato mayor al buscado. Y como la familia que estamos revisando es incremental, en el sentido que cada miembro puede construirse como el anterior y algo más, nuestra máquina siempre podrá rectificarse al notar su error.

Así, por teorema de invarianza, esta máquina nos muestra que

$$I^\infty\left(\Phi_{\mathcal{P}}[\text{máx}\{m : Q^\infty(m) \leq x\}]\right) \leq I^\infty\left(\{0; \ldots; |x_{(2)}| - 1\}\right) + |x_{(2)}| + c_{\mathfrak{M}^\infty_{\alpha\,\mathcal{P}}}$$

que es lo que queríamos demostrar (con $c_{\Phi_{\mathcal{P}}} = c_{\mathfrak{M}^\infty_{\alpha\,\mathcal{P}}}$). *Q.E.D.*

Recordemos que si $A \in \Phi_{\mathcal{P}}$, $\Phi_{\mathcal{P}}[\#A] = A$. Por otro lado, por lema 37, sabemos que, si A es finito y $P^\infty(A) > 2^{-n}$, existe un racional x_A de la forma $k/2^n$ en el intervalo $(Q^\infty(\#A); Q^\infty(\#A + 1))$.

Entonces, si $A \in \Phi_{\mathcal{P}}$, $Phi_{\mathcal{P}}[\text{máx}\{m : Q^\infty(m) \leq x_A\}] = A$. Luego, podríamos utilizar ese racional x_A en nuestra máquina $\mathfrak{M}^\infty_{\alpha\,\mathcal{P}}$ para obtener el conjunto A.

Como la representación binaria de este x_A sólo ocupa n *bits*, podemos concluir que:

$$\text{Si } A \in \Phi_{\mathcal{P}} \text{ y } P^\infty(A) > 2^{-n}$$
$$I^\infty(A) \leq n + I^\infty\left(\{0; \ldots; n-1\}\right) + c_{\mathfrak{M}^\infty_{\alpha\,\mathcal{P}}}$$

La proposición 28 nos mostró que $I^\infty\left(\{0; \ldots; n-1\}\right) = O(\log n)$. Podemos, entonces reescribir lo anterior como:

$$\text{Si } A \in \Phi_{\mathcal{P}} \text{ y } P^\infty(A) > 2^{-n},\ I^\infty(A) \leq n + O(\log n)$$

Pero decir que $P^\infty(A) > 2^{-n}$ es lo mismo que decir que $H^\infty(A) < n$. Entonces, tomando $n = H^\infty(A) + 1$, obtenemos que:

$$\text{Si } A \in \Phi_{\mathcal{P}},\ I^\infty(A) \leq H^\infty(A) + O(\log H^\infty(A))$$

que es lo que queríamos demostrar. *Q.E.D.*

5.3.2. Sobre la codificación en las familias consecutivas

Dedicaremos esta sección a mostrar que el teorema de codificación que pudimos verificar para las familias uniformes, no vale para familias consecutivas.

Teorema 39 *Para toda familia incremental Φ_{P_c}* NO *es cierto que si $A \in \Phi_{P_c}$*

$$I^\infty(A) = H^\infty(A) + O(1)$$

Demostración. Supongamos que el teorema de la codificación se cumple sobre las familias consecutivas. Entonces debería valer que, para todo $A \in \Phi_{P_c}$

$$I^\infty(A) \leq H^\infty(A) + c$$

donde c es un valor que no depende del conjunto.

Luego, para cada uno de estos conjuntos que tuviera su entropía menor a un natural n, debería existir un programa minimal de menos de $n + c$ *bits*. Esto implica que

$$\#\{A \in \Phi_{P_c} : H^\infty(A) < n\} \leq \#\{A \in \Phi_{P_c} : I^\infty(A) < n + c\}$$

Veamos los siguientes resultados, cuyas demostraciones presentaremos más adelante.

Corolario 40

$$\#\{A \in \Phi_{P_c} : I^\infty(A) < n\} \leq 2^{n - I^\infty(\{0;\ldots;n-1\}) + O(\log I^\infty(\{0;\ldots;n-1\}))}$$

Teorema 41

$$\#\{A \in \Phi_{P_c} : H^\infty(A) < n\} \geq 2^{n - I^{\infty\prime}(\{n\}) + O(1)}$$

Utilizando estos resultados junto con la inecuacón que veníamos manipulando, debería cumplirse que

$$n - I^{\infty\prime}(\{n\}) + O(1) \leq n + c - I^\infty(\{0;\ldots;n+c-1\}) + O(\log(I^\infty(\{0;\ldots;n+c-1\})))$$

Sabemos (ver [2, teo. 7(a)]) que el tamaño de las especificaciones de dos naturales no pueden tener una diferencia de tamaño mayor que el logaritmo de la diferencia entre los naturales que especifican. Por ello, la desigualdad anterior puede agrandarse, a lo sumo, a

$$n - I^{\infty\prime}(\{n\}) + O(1) \leq n - I^\infty(\{0;\ldots;n-1\}) + O(\log(I^\infty(\{0;\ldots;n-1\}))) + c$$

Resumiendo: si valiera el teorema de la codificación para fammnilias consecutivas, debería valer, para todo n natural, que

$$n + I^\infty(\{0;\ldots;n-1\}) + O(1) \leq n + I^{\infty\prime}(\{n\}) + O(\log(I^\infty(\{0;\ldots;n-1\}))) + c$$

Sin embargo, por teorema 32.2 sabemos que, dado un natural k, el menor m tal que $m > 2^k$ y $I^\infty(\{0;\ldots;m-1\}) > k$, cumple que $I^\infty(\{0;\ldots;m-1\}) \leq k + O(\log k)$ y $I^{\infty\prime}(\{m\}) \leq \log k + O(\log\log k)$. Entonces, ese m debería cumplir que

$$m + k + O(1) \leq m + \log k + O(\log\log k) + O(\log(k + \log k))$$

Teniendo en mente que k no es otra cosa que $\lfloor \log m \rfloor$, notemos que debe existir un m_0 a partir del cual la desigualdad anterior deja de valer, ya que el miembro izquierdo crece más rápidamente que el derecho a medida que incrementamos m.

Este absurdo ha surgido de suponer que el teorema de la codificación se verificaba sobre las familias consecutivas. *Q.E.D.*

Demostración del corolario 40

Demostraremos el corolario 40 utilizando que dicho resultado es un caso particular del siguiente teorema.

Teorema 42 (Chaitin)

$$\#\{A : I^{\infty}(A) < n\} \leq 2^{n - I^{\infty}(\{0;\ldots;n-1\}) + O(\log I^{\infty}(\{0;\ldots;n-1\}))}$$

Demostración. Para ver cuántos conjuntos tienen programas mínimos de menos de n dígitos, nuestra atención debe volcarse hacia la cantidad de *bits* que lee nuestra máquina universal $\mathfrak{U}^{\infty}$ con cada programa.

Lema 43 (Chaitin) *Existe una constante c tal que*

$$\#\{A : I^{\infty}(A) = m\} \leq P^{\infty}(\{0;\ldots;m-1\}) \cdot 2^{m+c}$$

Sea $\mathfrak{M}^{\infty}_{\mathtt{input}}$ la máquina que simula el comportamiento de nuestra máquina universal $\mathfrak{U}^{\infty}$ pero, en vez de imprimir lo que ella imprimiría, cuando lee el i-ésimo dígito de su cinta fuente, imprime el natural i en su cinta destino.

Así, si $\mathfrak{p}$ es un programa para $\mathfrak{U}^{\infty}$,

$$\mathfrak{M}^{\infty}_{\mathtt{input}}(\mathfrak{p}) = \{0;\ldots;|\mathfrak{p}|-1\}$$

Por otro lado, cada programa minimal de tamaño m, provoca que se lean m de sus *bits*. Luego, por cada conjunto que tiene un programa minimal de tamaño m, hay un aporte de 2^{-m} en la probabilidad de obtener el consecutivo $\{0;\ldots;m-1\}$ en nuestra máquina $\mathfrak{M}^{\infty}_{\mathtt{input}}$.

$$\#\{A : I^{\infty}(A) = m\} \cdot 2^{-m} \leq P_{\mathfrak{M}^{\infty}_{\mathtt{input}}}(\{0;\ldots;m-1\})$$

Finalmente, por teorema de invarianza (teorema 27)

$$\#\{A : I^{\infty}(A) = m\} \leq P^{\infty}(\{0;\ldots;m-1\}) \cdot 2^{m + c_{\mathfrak{M}^{\infty}_{\mathtt{input}}}}$$

Esto completa la demostración de nuestro lema (con $c = c_{\mathfrak{M}^{\infty}_{\mathtt{input}}}$).*Q.E.D.*

Así hemos hallado una cota para la cantidad de conjuntos que tienen programas mínimos de una longitud dada. Parece un buen primer paso, pero nuestro objetivo es hallar una cota para la cantidad de conjuntos que tengan programas minimales menores a cierta longitud.

Dado nuestro primer paso, arriesguemos el segundo paso de forma obvia:

$$\begin{array}{rcl} \#\{A : I^{\infty}(A) < n\} & = & \sum_{i=0}^{n-1} \#\{A : I^{\infty}(A) = i\} \\ & \leq & \sum_{i=0}^{n-1} P^{\infty}(\{0;\ldots;i-1\}) \cdot 2^{i+c} \end{array}$$

Para formar esta cota estamos sumando probabilidades de obtener consecutivos que guardan una relación bastante fuerte: son todos los consecutivos "menores a n". ¿De qué forma podremos aprovechar esta relación?

En ese sentido si n es un natural cualquiera, y l otro natural menor a n, ¿cuánto más grande que la probabilidad de obtener el consecutivo $\{0;\ldots;n-1\}$ podrá ser la probabilidad de obtener el consecutivo $\{0;\ldots;l-1\}$?

El siguiente lema responderá nuestra pregunta.

Lema 44 (Chaitin) *Existe una constante d tal que, para todo par de naturales n y l tal que $n > l$, se cumple que*

$$P^{\infty}(\{0;\ldots;n-1\}) \cdot (n-l)^2 \cdot d \geq P^{\infty}(\{0;\ldots;l-1\})$$

Demostración. Para la demostración de este lema necesitaremos la definición que presentamos a continuación.

Definición 38 (Join) *Sean A y B dos conjuntos de naturales. Llamaremos $A \oplus B$ al conjunto formado de la siguiente manera:*

$$A \oplus B = \{2a : a \in A\} \cup \{2b+1 : b \in B\}$$

Llamemos $\mathfrak{M}^{\infty}_{\oplus}$ a la máquina tal que por cada programa $\mathfrak{p}$ que provocaría que $\mathfrak{U}^{\infty}$ imprima el conjunto $A \oplus B$, imprime el consecutivo $\{0;\ldots;c-1\}$ donde $c = (\text{máx}\{a : a \in A\} + \text{máx}\{b : b \in B\})$.

Para que $\mathfrak{M}^{\infty}_{\oplus}$ pueda llevar a cabo esta tarea, basta con que, cada vez que $\mathfrak{U}^{\infty}$ imprimiría un elemento de uno de los conjuntos, nuestra máquina imprima todos los valores menores a la suma entre el mencionado elemento y el último elemento del otro conjunto que $\mathfrak{U}^{\infty}$ hubiese imprimido.

Entonces

$$P_{\mathfrak{M}^{\infty}_{\oplus}}(\{0;\ldots;n-1\}) \geq P^{\infty}(\{0;\ldots;l-1\} \oplus \{0;\ldots;n-l-1\})$$

Y, luego, por teorema de invarianza (teorema 27),

$$P^{\infty}(\{0;\ldots;n-1\}) \cdot 2^{c_{\mathfrak{M}^{\infty}_{\oplus}}} \geq P^{\infty}(\{0;\ldots;l-1\} \oplus \{0;\ldots;n-l-1\})$$

Por otro lado, por [2, Teo 3(f), pág. 6], sabemos que

$$P^{\infty}(A \oplus B) \succeq P^{\infty}(A) \cdot P^{\infty}(B)$$

Lo que implica que

$$P^{\infty}(\{0;\ldots;n-1\}) \cdot 2^{c_{\mathfrak{M}^{\infty}_{\oplus}}} \geq P^{\infty}(\{0;\ldots;l-1\}) \cdot P^{\infty}(\{0;\ldots;n-l-1\}) \quad (5.4)$$

Por otro lado, como vimos en la Proposición 7, $P^{\infty}(A) \geq 2^{-I^{\infty}(A)}$. Además, gracias a la representación autodelimitante de un número (proposición 28) sabemos que $2^{-I^{\infty}(\{0;\ldots;m-1\})} \geq 2^{-(\log m + 2\log\log m + c_{\sigma})}$. Entonces

$$P^{\infty}(\{0;\ldots;m-1\}) \geq \frac{2^{-(c_{\sigma}+1)}}{m \log m}$$

Finalmente,

$$P^{\infty}(\{0;\ldots;m-1\}) \geq \frac{1}{m^2 \cdot 2^{c_{\sigma}+1}} \quad (5.5)$$

Juntando (5.4) y reemplazando m en (5.5) por $n-l$, obtenemos

$$P^\infty(\{0;\ldots;n-1\})(n-l)^2 \cdot 2^{c_{\mathfrak{m}^\infty_\oplus}+c_\sigma+1} \geq P^\infty(\{0;\ldots;l-1\})$$

Que es lo que queríamos demostrar, con $d = 2^{c_{\mathfrak{m}^\infty_\oplus}+c_\sigma+1}$. *Q.E.D.*

El lema que acabamos de ver nos puede servir para tener una idea de la cota que veníamos manejando.

$$\begin{array}{rcl} \sum_{i=0}^{n-1} P^\infty(\{0;\ldots;i-1\}) & \leq & \sum_{j=1}^{n}(n-j)^2 \cdot P^\infty(\{0;\ldots;n-1\}) \cdot d \\ & = & d \cdot P^\infty(\{0;\ldots;n-1\}) \sum_{j=1}^{n}(n-j)^2 \end{array}$$

Como vemos, si bien la suma de las probabilidades de los consecutivos menores a n podría crecer más allá de la probabilidad de obtener el consecutivo n-ésimo, este crecimiento es sólo polinomial respecto de n. Veamos cómo encaja esto con la cota que nos interesa.

$$\begin{array}{rcl} \#\{A : I^\infty(A) < n\} & \leq & \sum_{i=0}^{n-1} 2^{i+c} \cdot P^\infty(\{0;\ldots;i-1\}) \\ & \leq & d \cdot 2^c \cdot P^\infty(\{0;\ldots;n-1\}) \sum_{i=0}^{n-1} 2^i (n-i)^2 \end{array}$$

Si bien este resultado no parece muy prometedor, ya que el factor que acompaña a la probabilidad de obtener el consecutivo n-ésimo es ahora más grande que antes (su crecimiento es exponencial respecto de n), notemos lo siguiente: tener una potencia de 2 en cada término de esa sumatoria es lo mismo que multiplicar toda la suma por la mayor de ellas y dividir cada sumando por la potencia correspondiente a ese término de la suma. Podemos lograr esto simplemente aplicando el cambio de variable: $k = n - i$.

$$\begin{array}{rcl} \#\{A : I^\infty(A) < n\} & \leq & d \cdot 2^c \cdot P^\infty(\{0;\ldots;n-1\}) \sum_{k=1}^{n} 2^{(n-k)} k^2 \\ & = & d \cdot 2^{c+n} \cdot P^\infty(\{0;\ldots;n-1\}) \sum_{k=1}^{n} 2^{-k} k^2 \end{array}$$

Pero la serie $\sum_{k=1}^{\infty} 2^{-k}k^2$ converge. Entonces, existe una constante $\tilde{c}$ tal que $\tilde{c} \geq d \cdot 2^c \cdot \sum_{k=1}^{n} 2^{-k}k^2$. Por lo tanto,

$$\#\{A : I^\infty(A) < n\} \leq \tilde{c} \cdot 2^n \cdot P^\infty(\{0;\ldots;n-1\}) \tag{5.6}$$

Por otro lado, por teorema 36 y como los consecutivos de forma $\{0;\cdots;n-1\}$ son una familia consecutiva, sabemos que:

$$I^\infty(\{0;\ldots;n-1\}) \leq H^\infty(\{0;\ldots;n-1\}) + O(\log H^\infty(\{0;\ldots;n-1\}))$$

Lo que, aprovechando que $H^\infty(A) \leq I^\infty(A)$, fácilmente puede llevarse a

$$P^\infty(\{0;\ldots;n-1\}) \leq 2^{-I^\infty(\{0;\ldots;n-1\})+O(\log I^\infty(\{0;\ldots;n-1\}))}$$

Utilizando este último resultado junto con (5.6), obtenemos la cota que estábamos buscando:

$$\#\{A : I^\infty(A) < n\} \leq 2^{n-I^\infty(\{0;\ldots;n-1\})+O(\log I^\infty(\{0;\ldots;n-1\}))}$$

Se completa así la demostración del teorema 42. *Q.E.D.*

Demostración del teorema 41

Para demostrar el teorema 41 comenzaremos por ver el siguiente resultado.

Teorema 45 *Sea $\Phi_{P_\mathcal{C}}$ una familia consecutiva. Existe una máquina $\mathfrak{M}^{\infty}_{--\rightarrow}$ tal que, por cada programa p que provoca que $\mathfrak{U}^{\infty\prime}$ imprima el conjunto $\{k\}$, toda secuencia que comience con esa cadena p, escrita en la cinta de entrada de $\mathfrak{M}^{\infty}_{--\rightarrow}$, provoca como resultado el consecutivo*

$$\Phi_{P_\mathcal{C}}[2^t + k]$$

donde t es el máximo entre el número de pasos que le toma a $\mathfrak{U}^{\infty\prime}(p)$ imprimir $\{k\}$ y, de todos los programas por los que $\mathfrak{U}^{\infty\prime}$ consulta al oráculo y se detienen, el número de pasos que tarda en detenerse el que se detiene último.

Demostración. Con este teorema trataremos de ver cómo podemos utilizar el poder de cómputo de las máquinas sin oráculo para interpretar la información condensada en un programa para cómputadoras con oráculo.

Si el tiempo es un camino, con una baldosa por instante, el oráculo es un viejo con ojos de lince que llega a ver el final de ese camino. Pero, aún miopes de tiempo como son las máquinas sin oráculo, nos pueden brindar el resultado que enuncia este teorema.

Si bien las máquinas convencionales no pueden resolver exactamente los problemas que pueden resolver las que disponen de un oráculo, pueden simular su trabajo utilizando un oráculo "a cuerda". Mediante un proceso de "dovetailing" podemos confeccionar un *oráculo restringido* que, dado un programa, nos diga si el cómputo que éste provoca se detiene en t o menos pasos.

Dicho oráculo restringido siempre va a asumir que los procesos de cómputo no se detienen hasta que pueda demostrarse lo contrario.

Debido a esta particularidad, consultado por un programa p, el oráculo restringido podría contestar "no" cuando el oráculo verdadero respondería "sí"; pero debemos tener en cuenta que no estamos formulando la misma pregunta, y es por eso que no obtenemos la misma respuesta. Al oráculo restringido le estamos preguntando si p provoca un cómputo que se detiene en t o menos pasos, mientras que al oráculo verdadero le estamos preguntando directamente si p provoca un cómputo que se detiene en algún momento, es decir, sin restringir el tiempo.

Podríamos, entonces, construir una máquina de cómputos eternos $\mathfrak{M}^{\infty}_{--\rightarrow}$ que simule el comportamiento de una máquina universal con oráculo $\mathfrak{U}^{\infty\prime}$ de manera que cada vez que ésta consultase a su oráculo, $\mathfrak{M}^{\infty}_{--\rightarrow}$ consultará a un oráculo restringido.

Al hallar una secuencia $\mathfrak{p}$ en su cinta de entrada, $\mathfrak{M}^{\infty}_{--\rightarrow}$ irá simulando pasos de cómputo de $\mathfrak{U}^{\infty\prime}$. Cuando se simulen t pasos de cómputo de $\mathfrak{U}^{\infty\prime}$, se utilizará el oráculo restringido al tiempo t.

Así, en algún momento (llamémoslo t), $\mathfrak{M}^{\infty}_{--\rightarrow}$ podría estar dispuesta a imprimir un número (llamémoslo k) en su cinta de salida; pero, ¿qué relación guardará este número con el resultado que se obtiene al escribir $\mathfrak{p}$ en la cinta de entrada de $\mathfrak{U}^{\infty\prime}$?

Si, durante la simulación que resultó en la impresión de k, $\mathfrak{U}^{\infty\prime}$ no usó su oráculo, o los procesos generados por los programas por los que consultó al oráculo se detuvieron antes de los t pasos de cómputo o bien no se van a detener nunca, el oráculo restringido a t pasos habrá dado las mismas respuestas que el oráculo verdadero y entonces $k \in \mathfrak{U}^{\infty\prime}(p)$. Pero si alguno de ellos fuera a detenerse más allá de los primeros t pasos, el resultado de la simulación no tendría por qué guardar relación alguna con $\mathfrak{U}^{\infty\prime}(p)$.

A pesar de que el oráculo restringido no tiene el poder de un oráculo verdadero, ya que con el primero debemos esperar (toda la eternidad) por respuestas que el segundo puede darnos instantáneamente, una similitud de comportamiento une los destinos de ambos dispositivos. Esta similitud se materializa en la siguiente característica: a medida que tomamos t más grandes, las repuestas del oráculo restringido van a ser más acordes a las del verdadero.

En este sentido, más tarde o más temprano alcanzaremos un t lo suficientemente grande como para obtener las repuestas "correctas" del oráculo restringido.

Deberíamos, entonces, hallar una manera de imprimir k de alguna manera que nos permita rectificarnos más adelante si, con el paso del tiempo, descubrimos que k no formaba parte de $\mathfrak{U}^{\infty\prime}(p)$.

Una forma que ya hemos utilizado anteriormente en máquinas de cómputos eternos para rectificar errores es imprimir consecutivos. Pero esta manera sólo permite rectificarnos hacia resultados mayores, es decir, si en el momento t imprimiéramos $\Phi_{Pc}[k]$ y luego descubriéramos que k no era parte del resultado real, sólo podríamos subsanar el error si el verdadero número fuera mayor que k.

Pero notemos lo siguiente: cuando nuestra simulación indica que debe imprimirse k, no podemos saber si ese k sería imprimido o no por $\mathfrak{U}^{\infty\prime}$ pero, si la simulación tardó t pasos en obtener este resultado, k no puede ser mayor a $t-1$. Recordemos que, para imprimir k, estas máquinas deben imprimir 1, mover la cinta k lugares hacia la izquierda e imprimir otro 1.

Luego, si bien no nos va a servir de mucho imprimir el consecutivo $\Phi_{Pc}[k]$ podemos aprovechar la característica que acabamos de ver e imprimir un consectivo que tenga que ver con k y con el momento en el que nuestra simulación obtuvo aquel número, de manera tal que que si el verdadero k era más chico que algún k que hayamos encontrado antes, aún así lo que imprimamos sea más grande que lo que ya hayamos impreso.

Entonces, si nuestra simulación descubre el número k en el momento t, necesitamos imprimir el consecutivo $\Phi_{Pc}[f(k,t)]$ donde $f(k,t)$ debe cumplir que, cualquiera sea k, $f(k,t) < f(0,t+1)$.

Como ya vimos que si $\mathfrak{M}^{\infty}_{--\rightarrow}$ imprime k en el momento t, k es menor o igual a t, nos basta con

$$f(k,t) = 2^t + k$$

Finalmente, si p es un programa que provoca que $\mathfrak{U}^{\infty\prime}$ imprima el unitario $\{k\}$, cualquier secuencia que comience con la cadena p escrita en la cinta de entrada de $\mathfrak{M}^{\infty}_{--\rightarrow}$ provocará que dicha máquina genere el consecutivo $\Phi_{Pc}[2^t + k]$

donde t es el máximo entre el número de pasos que le toma a $\mathfrak{U}^{\infty\prime}(p)$ imprimir $\{k\}$ y, de todos los programas por los que $\mathfrak{U}^{\infty\prime}$ consulta al oráculo y se detienen, el número de pasos que tarda en detenerse el que se detiene último. *Q.E.D.*

Corolario 46 *Sea $\Phi_{P_\mathcal{C}}$ una familia consecutiva. Por cada conjunto unitario $\{k\}$ existe un l_k diferente tal que*

$$P^{\infty}(\Phi_{P_\mathcal{C}}[l_k]) \;\geq\; P^{\infty\prime}(\{k\})$$

Demostración. Gracias a la optimalidad de las máquinas universales, podemos ver que, por cada programa p tal que $\mathfrak{M}^{\infty}_{\dashrightarrow}(p) = \Phi_{P_\mathcal{C}}[2^t + k]$, existe un programa q tal que $\mathfrak{U}^{\infty}(q) = \Phi_{P_\mathcal{C}}[2^t + k]$.

Luego, por el teorema 45, la probabilidad de que un programa generado por tiradas de monedas provoque que $\mathfrak{U}^{\infty}$ enumere el conjunto $\Phi_{P_\mathcal{C}}[2^t + k]$ es, por lo menos, como la probabilidad de que un programa generado por tiradas de monedas provoque que $\mathfrak{U}^{\infty\prime}$ enumere al conjunto $\{k\}$, que es lo que queríamos demostrar con $l_k = 2^t + k$. *Q.E.D.*

Antes de seguir, recordemos que los programas q de los que hablábamos más arriba, son de la forma $r \cdot p$ donde r es un prefijo $0^g 1$ que provoca que la máquina $\mathfrak{U}^{\infty}$ se comporte como $\mathfrak{M}^{\infty}_{\dashrightarrow}$.

Luego, por cada unitario $\{k\}$ tal que $I^{\infty\prime}(\{k\}) < n - c$, donde $c = |r| = g + 1$, y por lo tanto $c > 0$, resulta que,

$$2^{-I^{\infty\prime}(\{k\})} > 2^{c-n} > 2^{-n}$$

Uniendo esto con el resultado del corolario anterior y utilizando que, por el teorema de invarianza sabemos que $P_{\mathfrak{U}\prime}(\{k\}) \;\geq\; 2^{-I^{\infty\prime}(\{k\})}$, obtenemos que el l_k correspondiente es tal que

$$P^{\infty}(\Phi_{P_\mathcal{C}}[l_k]) \;\geq\; P^{\infty\prime}(\{k\}) \geq 2^{-I^{\infty\prime}(\{k\})} > 2^{-n}$$

Lo que nos conduce trivialmente, utilizando la definición de $H^{\infty}(A)$, al siguiente resultado.

Corolario 47 *Sea $\Phi_{P_\mathcal{C}}$ una familia consecutiva.*

$$\#\{C \in \Phi_{P_\mathcal{C}} : H^{\infty}(C) < n\} \geq \#\{\textit{unitarios } U : I^{\infty\prime}(U) \;<\; n - c\}$$

Para completar la demostración del teorema 41 necesitamos una cota inferior para la cantidad de conjuntos que tengan programas mínimos más cortos que cierta longitud.

Teorema 48 (Chaitin) *Existe una constante c tal que*

$$\#\{\textit{unitarios } U : I^{\infty}(U) \;<\; n - c\} \;\geq\; 2^{n-c-I^{\infty}(\{n-c\})+O(1)}$$

Demostración. Sea $\mathfrak{M}^{\infty}_{\frown}$ la máquina que dado un programa $\mathfrak{p}$ hace lo siguiente: simula el comportamiento de $\mathfrak{U}^{\infty}(\mathfrak{p})$ hasta que dicha máquina imprimiría el primer natural n, en ese momento lee todos los dígitos de su cinta de entrada que faltan leer para que sólo $n-1$ *bits* sean leídos. Finalmente, interpreta estos últimos *bits* leídos (llamémoslos $\mathfrak{s}$) como la representación binaria de un natural k, imprime el unitario $\{k\}$ y se va a dormir.

Podría ocurrir que, para imprimir el primer natural, se deban leer más de $n-1$ dígitos. En ese caso $\mathfrak{M}^{\infty}_{\frown}$ se va a dormir sin imprimir nada (en penitencia). Pero, si la primer parte de $\mathfrak{p}$ es un programa minimal para el unitario $\{n\}$, con todos sus *bits* superfluos eliminados, seguramente nos quedará espacio para escribir $\mathfrak{s}$, la representación binaria de un natural k.

Entonces, por cada elección de $\mathfrak{s}$ habrá un programa de longitud menor a n que imprime un unitario distinto. Luego, la cantidad de unitarios con I^{∞} menor a n debe ser mayor que la cantidad de elecciones posibles para $\mathfrak{s}$.

La cantidad de elecciones posibles para $\mathfrak{s}$ es

$$2^{|\mathfrak{s}|} = 2^{|\mathfrak{p}\cdot\mathfrak{s}|} - 2^{|\mathfrak{p}|} \geq 2^{n-1-I^{\infty}(\{n\})}$$

Entonces:

$$\#\{k : I_{\mathfrak{M}^{\infty}_{\frown}}(\{k\}) \ < \ m\} \ \geq \ 2^{m-I^{\infty}(\{m\})-1}$$

Como $I^{\infty}(A) \leq I_{\mathfrak{M}^{\infty}_{\frown}}(A) + c_{\mathfrak{M}^{\infty}_{\frown}}$, todos los conjuntos que cumplan que $I_{\mathfrak{M}^{\infty}_{\frown}}(A) < m$ cumplirán también que $I^{\infty}(A) < m + c_{\mathfrak{M}^{\infty}_{\frown}}$. Llamando $n = m + c_{\mathfrak{M}^{\infty}_{\frown}}$.

$$\#\{k : I^{\infty}(\{k\}) \ < \ n - c_{\mathfrak{M}^{\infty}_{\frown}}\} \ \geq \ 2^{n-c_{\mathfrak{M}^{\infty}_{\frown}}-I^{\infty}(\{n-c_{\mathfrak{M}^{\infty}_{\frown}}\})+O(1)}$$

que es el resultado que buscábamos (tomando $c = c_{\mathfrak{M}^{\infty}_{\frown}}$). *Q.E.D.*

Notemos que podemos reescribir el resultado anterior para el caso en el que trabajemos con máquinas con oráculo, de la siguiente manera:

$$\#\{\text{unitarios } U : I^{\infty\prime}(U) \ < \ n - c\} \ \geq \ 2^{n-c-I^{\infty\prime}(\{n-c\})+O(1)}$$

Finalmente, existe un prefijo de la forma 0^h1 que provoca que, para cada programa $\mathfrak{p}$, se cumpla que $\mathfrak{U}^{\infty}(0^h1\cdot\mathfrak{p}) \ = \ sub_c(\mathfrak{U}^{\infty}(\mathfrak{p}))$, donde sub_c es una función que toma un conjunto de naturales y le resta c a cada uno de sus elementos (es una función total –de modo que si a un número se le resta otro mayor, la operación da cero–, lo que impide que sea inyectiva).

Entonces, el programa mínimo que enumera $\{n-c\}$ debe ser a lo sumo tan grande como el que enumera a $\{n\}$ más el largo del prefijo 0^h1. Es decir, $I^{\infty\prime}(\{n\}) + h + 1 \geq \ I^{\infty\prime}(\{n-c\})$, o dicho de otro modo,

$$-I^{\infty\prime}(\{n-c\}) \ \geq \ -I^{\infty\prime}(\{n\}) \ + O(1)$$

Finalmente, juntando el corolario 47 con el equivalente para cómputos con oráculos del teorema 48, obtenemos que

$$\#\{\text{consecutivos } C : \ H^{\infty}(C) \ < \ n\} \ \geq 2^{n-I^{\infty\prime}(\{n\})+O(1)}$$

Con lo que completamos la demostración del teorema 41.

5.3.3. Un intento fallido

Dado que hemos visto que los programas de longitud minimal para conjuntos consecutivos no crece más allá del valor de su entropía más el logaritmo de ese valor, podríamos arriesgarnos a pensar que para otras familias de conjuntos finitos estructuralmente más complejas que las que acabamos de estudiar esta brecha se agranda aún más.

Podríamos, en ese sentido, tratar de aplicar una estrategia idéntica a la que aplicamos recién para conseguir un absurdo que nos indique que debe haber algún tipo de conjunto finito para el cual la mencionada brecha se incrementa más que para los consecutivos.

Deberíamos suponer, entonces, que para todo conjunto finito A vale que $I^\infty(A) \leq H^\infty(A) + O(\log H^\infty(A))$. Luego

$$\#\{\text{finitos } A : H^\infty(A) < n\} \leq \#\{\text{finitos } A : I^\infty(A) < n + O(\log n)\}$$

Usando que

$$2^{m - I^\infty{}'(\{m\}) + O(1)} \leq \#\{\text{finitos } A : H^\infty(A) < m\}$$

y que

$$\begin{array}{rcl} \#\{\text{finitos } A : I^\infty(A) < m\} & \leq & \#\{A : I^\infty(A) < m\} \\ & \leq & 2^{m - I^\infty(\{0;\ldots;m-1\}) + O(\log I^\infty(\{0;\ldots;m-1\}))} \end{array}$$

obtenemos que

$$\begin{array}{rcl} n - I^\infty{}'(\{n\}) + O(1) & \leq & n + O(\log n) - I^\infty(\{0;\ldots;n + O(\log n) - 1\}) + \\ & & + O(\log I^\infty(\{0;\ldots;n + O(\log n) - 1\})) \end{array}$$

Que es lo mismo que

$$\begin{array}{l} n + I^\infty(\{0;\ldots;n + O(\log n) - 1\}) + O(1) \leq \\ \leq n + O(\log n) + I^\infty{}'(\{n\}) + O(\log I^\infty(\{0;\ldots;n + O(\log n) - 1\})) \end{array}$$

Ya sabemos que la diferencia entre las especificaciones de dos naturales no puede ser mayor que el logaritmo de la diferencia de esos valores. Luego, debe valer que

$$I^\infty(\{0;\ldots;n-1\}) \leq I^\infty(\{0;\ldots;n + O(\log n) - 1\}) + O(\log\log n)$$

y que

$$I^\infty(\{0;\ldots;n + O(\log n) - 1\}) \leq I^\infty(\{0;\ldots;n-1\}) + O(\log\log n)$$

Entonces si se cumple la inecuación con que veníamos trabajando, debe cumplirse que

$$\begin{array}{l} n + I^\infty(\{0;\ldots;n-1\}) - O(\log\log n) + O(1) \leq \\ \leq n + O(\log n) + I^\infty{}'(\{n\}) + O(\log(I^\infty(\{0;\ldots;n + -1\}) + O(\log\log n))) \end{array}$$

Usando, como antes, que, dado k, el menor $m \geq 2^k$ tal que $I^\infty(\{0;\ldots;m-1\}) \geq k$ cumple que $I^\infty(\{0;\ldots;m-1\}) \leq k + O(\log k)$ y que $I^{\infty\prime}(\{m\}) \leq \log k + O(\log\log k)$, la inecuación anterior implica que

$$\begin{aligned} n + k + O(1) \quad \leq \quad & n + O(\log n) + \log k + O(\log\log k) + \\ & O(\log(k + O(\log k) + O(\log\log n))) + O(\log\log n) \end{aligned}$$

Pero el sumando $O(\log n)$ que aparece en el segundo término tiene tanto peso como el k (que no es otra cosa que $\lfloor \log n \rfloor$) del otro miembro. Por ello no podemos conseguir el absurdo que conseguimos para los consecutivos.

Aún más, con lo único que podemos "jugar" es con la diferencia entre una especificación de m y una de $m + O(\log m)$ que no puede ser más grande que $O(\log\log m)$, por lo que parece ser que esta estrategia ya nos brindó todo lo que nos podía dar.

Capítulo 6

Condiciones para la validez del teorema de invarianza

En este capítulo nos dedicaremos a establecer condiciones sobre familias de conjuntos de naturales, que sean suficientes para determinar la validez del teorema de codificación.

Estas condiciones surgirán de extender la versión del teorema de codificación para familias uniformes que vimos en el capítulo anterior.

6.1. Destilando la estrategia

En principio, notemos que la estrategia utilizada en la demostración del teorema de codificación para familias uniformes utiliza el resultado del mismo teorema para el caso de máquinas de cómputos finitos.

La clave de esta estrategia es poder hallar una conexión entre los objetos que estamos manejando con las máquinas de cómputos eternos y los que manejamos con las computadoras de cómputos finitos.

El descubrimiento de esta conexión puede verse como el surgimiento de un escalafón entre las familias de conjuntos de naturales. Aquellas que respondan a este requerimiento, sobre las que pueda establecerse esta conexión, son familias que no "aprovechan" el poder de cómputo de las máquinas eternas, ya que pueden ser manejadas por computadoras finitas.

Esta conexión se materializa en tres hechos. Primero, la posibilidad de asignar una palabra a cada familiar de la familia en cuestión.

Definición 39 (Representaciones canónicas finitarias) *Sea $\mathscr{A}$ una familia de conjuntos finitos de naturales, y $\mathbf{e} : \mathscr{A} \to 2^\star$ una función inyectiva total. Por cada conjunto $A \in \mathscr{A}$, llamaremos* representación canónica *de A a la cadena $\mathbf{e}(A)$.*

En segundo lugar, debemos poder "adivinar" a qué conjunto codifica una representación sólo con ver una parte finita de ella y, en el caso de que esta

predicción sea errónea, que el conjunto realmente representado no pertenezca a la familia.

Definición 40 (Adivinas) *Sea $\mathscr{A}$ una familia de conjuntos finitos de naturales, y $\mathbf{e} : \mathscr{A} \to 2^\star$ una función inyectiva total. LLamaremos* adivinas *a las fuciones $\mathbf{c_e} : 2^\star \to 2^\star$ tales que, para todo $d \in 2^\star$, si $\mathbf{c_e}(d) = e(A)$ no existe $c \in 2^\star$ tal que $\mathbf{c_e}(d \cdot c) = \mathbf{e}(B)$ con $A \neq B$.*

Estas capacidades "proféticas" que nos brinda el trabajo con este tipo de familias (sobre las que se pueden definir funciones *adivinas*) nos revelarán que la probabilidad de obtener alguno de sus familiares en una máquina eterna no es mayor que la probabilidad de obtener su correspondiente representación canónica en una computadora finita.

Lema 49 *Sean $\mathscr{A}$ una familia de conjuntos de naturales, $\mathbf{e} : \mathscr{A} \to 2^\star$ su función de representación canónica y $\mathbf{c_e} : 2^\star \to 2^\star$ la función* adivina *correspondiente. Si $\mathbf{c_e}$ es recursiva, existe un natural c tal que, para todo $A \in \mathscr{A}$,*

$$P^\infty(A) \leq P(\mathbf{e}(A)) \cdot 2^c$$

Demostración. Sea $\mathscr{M}_{\mathbf{c_e}}$ una máquina finita que, alimentada con el programa p, simula el comportamiento de la máquina eterna universal $\mathfrak{U}^\infty$ al ser alimentada con el mismo programa. Cuando ésta última imprimiría un elemento, llamémoslo s, del dominio de $\mathbf{c_e}$, $\mathscr{M}_{\mathbf{c_e}}$ imprime $\mathbf{c_e}(s)$ y termina.

Así, por cada programa p tal que $\mathfrak{U}^\infty(p) = A$, existe un prefijo $q \sqsubseteq p$ tal que $\mathscr{M}_{\mathbf{c_e}}(q) = \mathbf{e}(A)$. Entonces, para todo $A \in \mathscr{A}$,

$$P^\infty(A) \leq P_{\mathscr{M}_{\mathbf{c_e}}}(\mathbf{e}(A))$$

Luego, por teorema de invarianza (8), existe un natural $c_{\mathscr{M}_{\mathbf{c_e}}}$ tal que

$$P^\infty(A) \leq P(\mathbf{e}(A)) \cdot 2^{c_{\mathscr{M}_{\mathbf{c_e}}}}$$

que es lo que queríamos demostrar con $c = c_{\mathscr{M}_{\mathbf{c_e}}}$. *Q.E.D.*

Finalmente, debemos ser capaces, dada una representación canónica, de obtener el conjunto representado por ella. En verdad, nos alcanzará con obtener su representación correspondiente a la función $\mathfrak{v}$.

Definición 41 (Intérpretes) *Sean $\mathscr{A}$ una familia de conjuntos de naturales y $\mathbf{e} : \mathscr{A} \to 2^\star$ su función de representación canónica. Llamaremos* intérpretes *a las funciones $\mathbf{a} : 2^\star \to 2^\omega$ tales que, para todo $A \in \mathscr{A}$,*

$$\mathbf{a}(\mathbf{e}(A)) = d_A$$

donde $\mathfrak{v}(d_A) = A$.

Lema 50 *Sean $\mathscr{A}$ una familia de conjuntos de naturales, $\mathbf{e} : \mathscr{A} \to 2^\star$ su función de representación canónica y $\mathbf{a} : 2^\star \to 2^\omega$ su función* intérprete *asociada. Si $\mathbf{a}$ es recursiva, existe un natural $m_{\mathbf{e}}$ tal que, para todo $t \in Dom(\mathbf{a})$,*

$$I^\infty(A_t) \leq I(t) + m_{\mathbf{a}}$$

donde $A_t = \mathfrak{v}(\mathbf{a}(t))$, es decir, t es la representación canónica de A.

Demostración. Sea $\mathfrak{M}^\infty_{\mathbf{a}}$ una máquina eterna que, al ser alimentada con el programa p, simula el comportamiento de $\mathscr{U}(p)$, una máquina finita universal, y si esta imprimiera la cadena t, $\mathfrak{M}^\infty_{\mathbf{a}}$ imprimiría $\mathbf{a}(t)$. Entonces,

$$I^\infty_{\mathfrak{M}^\infty_{\mathbf{a}}}(\mathfrak{v}(\mathbf{a}(t))) \leq I(t)$$

Por teorema de invarianza, sabemos que existe un natural $m_{\mathbf{a}}$ tal que

$$I^\infty(A) \leq I^\infty_{\mathfrak{M}^\infty_{\mathbf{a}}}(A) + m_{\mathbf{a}}$$

Juntando ambos resultados obtenemos

$$I^\infty(\mathfrak{v}(\mathbf{a}(t))) \leq I(t) + m_{\mathbf{a}}$$

que es lo que queríamos demostrar. *Q.E.D.*

6.2. Aplicando la estrategia

Teorema 51 *Sea $\mathscr{A}$ una clase de conjuntos de naturales para la cual puedan definirse*

$\mathbf{e} : \mathscr{A} \to 2^\star$ *una función de representación canónica*

$\mathbf{c_e} : 2^\star \to 2^\star$ *una función adivina*

$\mathbf{a} : 2^\star \to 2^\omega$ *una función intérprete*

Si $\mathbf{c_e}$ y $\mathbf{e}$ son recursivas, existe un natural $c_{\mathscr{A}}$ tal que, para todo $A \in \mathscr{A}$,

$$I^\infty(A) \leq H^\infty(A) + c_{\mathscr{A}}$$

Demostración. Por el lema 49, sabemos para todo $A \in \mathscr{A}$,

$$P^\infty(A) \leq P(\mathbf{e}(A)) \cdot 2^c$$

Luego, por el teorema 10, existe una constante m tal que

$$P^\infty(A) \leq 2^{-I(\mathbf{e}(A))+m+c}$$

Finalmente, por el lema 50, sabemos que existe una constante $m_{\mathbf{e}}$ tal que

$$P^\infty(A) \leq 2^{-I^\infty(A)+m+c+m_{\mathbf{e}}}$$

Que es lo mismo que decir que

$$I^\infty(A) \leq H^\infty(A) + m + c + m_{\mathbf{e}}$$

Q.E.D.

Bibliografía

[1] A. M. Turing: *On Computable Numbers, with an Application to the Entscheidungsproblem*, ?? (1936), pp. 230-265.

[2] G. Chaitin: *Algorithmic Entropy of Sets*, Computers & Mathematics with Applications 2 (1976), pp. 233-245.

[3] G. Chaitin: *A Theory Of Program Size Formally Identical To Information Theory*, Journal of the ACM 22 (1975), pp. 320-340.

[4] D. Castro, M. Giusti, J. Heintz,G. Matera, L. M. Pardo: *The Hardness of Polynomial Equation Solving*, Foundations of Computational Mathematics (2003), OF1-OF74.