UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Matemática

Tesis de Licenciatura

Conjuntos K-Triviales y Complejidad Computacional

Ezequiel Erbaro

Director: Dr. Santiago Figueira

2 de Julio de 2012

Índice general

1. Introducción **1**
1.1. Organización 3

2. Preliminares **5**
2.1. Notación y convenciones 5
2.2. Conceptos básicos de Computabilidad 6
2.2.1. Reducibilidad de Turing 9
2.2.2. El Operador de Salto y relativización 10
2.2.3. Aproximaciones de funcionales y principio del uso 11
2.2.4. Conjuntos Δ_2^0, n-c.e. y ω-c.e 11
2.2.5. Árboles, caminos y clases Π_1^0 y Σ_1^0 12
2.3. Complejidad de Kolmogorov 13
2.3.1. Complejidad Simple de Kolmogorov 14
2.3.2. Máquinas libres de prefijos y complejidad 17
2.4. Aleatoriedad de los reales 20
2.4.1. El enfoque del programador 21
2.4.2. El enfoque estadístico 22

3. Jump Traceability **23**
3.1. Conjuntos superlow y jump-traceables 24
3.1.1. Todo conjunto n-c.e. y jump traceable es superlow 24
3.1.2. Todo conjunto n-c.e y superlow es jump traceable 26

4. Strong Jump Traceability **31**
4.1. Existencia y propiedades básicas 31
4.2. Aproximaciones del salto 33
4.3. Trazabilidad y complejidad simple de Kolmogorov 35

5. K-Trivialidad **39**
5.1. C-Trivialidad 39
5.2. Conjuntos K-Triviales 40
5.2.1. Construyendo un conjunto K-Trivial no computable 41
5.3. Subclases de los conjuntos K-triviales 43

5.3.1. Un conjunto K-trivial y c.e. que no es strongly jump-traceable . . . 44
5.3.2. Los conjuntos c.e. y strongly jump-traceables son K-triviales 47

Conjetura **55**

Agradecimientos

A mi papá, sin su ayuda, paciencia y comprensión nada de esto hubiese sido posible: ni la tesis ni la carrera.

A mi familia, que siempre me alentó a seguir y estuvo al lado mío durante todo la carrera.

A Santiago, por cada una de las explicaciones, por enseñarme con paciencia y responder mis preguntas: las obvias y las difíciles.

A mis primeros compañeros y amigos: Fede, Ro, Mari, Tata, Lucio y Javi, a todos los que vinieron después, a los muchachos de Dale Chechu, a Adrián que siempre me prestó sus carpetas y parciales.

A cada uno de mis compañeros de cursada que me dieron una mano con lo que no entendía, a los ayudantes y profesores que en los parciales veían el vaso medio lleno.

A mis amigos de toda la vida, y a mis amigos de salsa: no se puede hacer una carrera sin distraerse, sin divertirse.

A todos, gracias.

Capítulo 1

Introducción

En la presente tesis estudiaremos la interacción de ciertos aspectos particulares entre complejidad computacional y aleatoriedad. Ambas ramas, a lo largo del tiempo, se han visto enriquecidas mutuamente a través de desarrollos independientes y paralelos, resultando en algunas ocasiones equivalentes. Estas coincidencias han sucitado investigaciones recientes que tienen por objeto determinar hasta qué punto se pueden caracterizar a través de una teoría propiedades de la otra.

El primer resultado obtenido dentro del presente marco de investigación consiste en una caracterización debida a Loveland de los conjuntos computables a través de la complejidad plana de Kolmogorov. Informalmente podemos definir la complejidad plana de Kolmogorov de un objeto como una medida de los recursos necesarios para especificarlo. Notaremos en lo que sigue a la complejidad plana de Kolmogorov con la letra C. Loveland [18] demostró que un real α es computable si y sólo si la complejidad de la secuencia de los primeros n bits de α dado n como información, es decir $C(\alpha \upharpoonright n|n)$, es acotada. Podemos, en consecuencia, caracterizar la noción de ser computable con el hecho de tener segmentos iniciales de baja complejidad. El resulado de Loveland fue posteriormente extendido por Chaitin [4] quien demostró que un real α es computable si y sólo si $C(\alpha \upharpoonright n) \leq C(n) + O(1)$.

La introducción de la complejidad de Kolmogorov libre de prefijo debida a Levin [16], Schnorr [30] y Chaitin [3], notada en adelante con la letra K, sucitó de manera análoga al caso de complejidad plana, la siguiente pregunta: Implica una cota constante en $K(\alpha \upharpoonright n) - K(n)$ que α es computable como en el caso del teorema de Chaitin? Este último logró demostrar únicamente que si existía tal acotación entonces $\alpha \in \Delta^0_2$, donde Δ^0_2 son aquellos conjuntos Turing reducibles al *Halting Problem*. Un resultado fundamental de Solovay [34] contesta negativamente la pregunta: mediante un complejo argumento demuestra la existencia de un real α no computable y Δ^0_2 que satisface

$$\forall n \; K(\alpha \upharpoonright n) \leq K(n) + b. \tag{1.1}$$

Mediante una variación del argumento de Solovay se obtienen luego demostraciones más sencillas debidas a Calude y Coles [2], y posteriormente Downey, Hirschfeldt, Nies y Stephan [6] consiguen, utilizando el método de funciones de costo, que el real obtenido sea computablemente enumerable (c.e.)

Reales α que satisfacen (1.1) son llamados *K-Triviales* [6] y serán estudiados en detalle en el capítulo final de la tesis.

En paralelo al estudio de los K-Triviales se introdujo el concepto de ser *bajo* para una clase de complejidad $\mathcal{C}$, donde $\mathcal{C}$ constituye una clase de conjuntos que comparten una propiedad de complejidad particular, por ejemplo ser computable. Dado $\mathcal{C}$ muchas veces resulta natural considerar su versión relativizada $\mathcal{C}^A$, de esta forma decimos que un conjunto A es *bajo para* $\mathcal{C}$ cuando $\mathcal{C}^A = \mathcal{C}$, es decir: el poder de A como oráculo es tan bajo que no expande nuestra clase $\mathcal{C}$. Dos ejemplos de clases de conjuntos bajos son: los conjuntos *bajos para aleatoriedad de Martin-Löf* y los conjuntos *bajos para* K. La primer clase de conjuntos puede describirse como aquellos conjuntos A tales que la colección de reales A-aleatorios coincide con los reales 1-aleatorios. El concepto de ser bajo para K por el otro lado fue introducido por An. A. Muchnik quien logró construir un conjunto A tal que existe un $b \in \mathbb{N}$ tal que para toda toda cadena σ se satisface

$$K^A(\sigma) \geq K(\sigma) - b.$$

O sea, salvo constante, el oráculo A no logra superar al oráculo vacío a la hora de hallar una descripción sucinta de la cadena σ.

Se hizo evidente pronto la semejanza entre la construcción de un real K-Trivial mediante funciones de costo y la construcción de un real bajo para Martin-Löf debida a Kŭcera y Terwijn [14]. El resultado anterior no tardó en sucitar la siguiente pregunta: Coinciden acaso las clases de los K-Triviales, los conjuntos bajos para K y los bajos para Martin-Löf? La respuesta es afirmativa. Este resultado es uno de los más importantes y recientes de la teoría y fue demostrado por Nies [7] y por Hirschfeldt y Nies [26]. Posteriormente se demostró en [10] que la clase de los conjuntos K-Triviales coincidía también con la clase de las *bases para aleatoriedad de Martin-Löf*: es decir conjuntos A tales que existe un real B A-aleatorio tal que $A \leq_T B$. Para una introducción accesible de los resultados anteriores puede consultarse [23].

Antes de enunciar el resultado fundamental estudiado en esta tesis debemos introducir ciertos conceptos. Diremos que una función $h : \mathbb{N} \mapsto \mathbb{N}$ es un *orden* si es computable, no decreciente y tiende a infinito. Una traza c.e. es una familia uniformemente c.e. $\{T_x\}$ de conjuntos finitos. Diremos que $\{T_x\}$ *traza* a la función parcial f si para todo x tal que $f(x) \downarrow$ tenemos que $f(x) \in T_x$ y además la traza *obedece* a la función de orden h, es decir $|T_x| \leq h(x)$. Por otro lado definimos el *salto* de A en e, es decir $J^A(e)$ como $\Phi_e^A(e)$. Un conjunto A se dirá *jump traceable* si su salto en el argumento e posee pocos valores. Más precisamente

Definición 1.0.1. Un conjunto A es *jump traceable* si existe una traza $\{T_x\}$ tal que

$$\forall e \; J^A(e) \downarrow \Rightarrow J^A(e) \in T_e$$

La introducción de este concepto por parte de Nies [25] permitió probar el siguiente resultado: Si A es K-Trivial entonces A es jump traceable. Podemos preguntarnos si vale la recíproca: son acaso los conjuntos jump traceables K-triviales? Puede verse que no.

Investigaciones posteriores centraron su esfuerzo en hallar una propiedad que perteneciese exclusivamente a la teoría de la computabilidad, es decir que no involucrase ningún concepto de aleatoriedad y fuese equivalente a ser K-Trivial.

Figueira, Nies y Stephan [9] consideraron una noción que generaliza a la de ser jump-traceable que resultó fructífera en tanto permitió hallar una subclase propia de los conjuntos K-triviales utilizando únicamente conceptos provenientes de la teoría de la computabilidad. Más precisamente introdujeron los conjuntos *Strongly jump traceble*

Definición 1.0.2. Un conjunto A se dirá *Strongly jump traceable* si J^A posee para cada función de orden h una traza T_e que obedece h

Podemos enunciar finalmente el teorema fundamental estudiado en esta tesis, debido a Cholak, Downey y Greenberg [5]

Teorema 1.0.3. *Los conjuntos c.e. y strongly jump traceble constituyen una clase propia de los K-triviales*

1.1. Organización

Luego de este capítulo introductorio, la tesis se dividirá en cuatro capítulos:

En el capítulo 1 nos centraremos en las nociones básicas de la teoría de computabilidad y complejidad descriptiva de cadenas que se requerirán a lo largo de la tesis. Se introducirán las nociones fundamentales de reducibilidad de Turing, el operador de salto y aproximaciones de funcionales. En lo que respecta a la parte de complejidad descriptiva se presentarán las nociones de complejidad plana de Kolmogorov y complejidad libre de prefijos, así como tambien sus propiedades básicas y teoremas que serán utilizados posteriormente, como el teorema de Kraft-Chaitin.

En el capítulo 2 introduciremos la noción fundamental de *Jump Traceability*, sus propiedades básicas y su relación con los conjuntos *superlow*. El teorema fundamental de este capítulo será la coincidencia de la noción de superlow con jump traceable para conjuntos n-c.e. Al finalizar el capítulo exploraremos posibles extensiones de este teorema.

En el capítulo 3 se estudiará un generalización del concepto de ser jump traceable que puede encontrarse en [8] y se denomina *Strong jump traceability*. Se probará la existencia de estos conjuntos, sus propiedades elementales y finalmente se demostrará una caracterización de los conjuntos strongly jump traceables que involucra la complejidad plana de Kolmogorov.

En el capítulo 4 se estudiará la noción fundamental de *K-trivialidad*, se verá porqué resulta central para la teoría y de qué manera se relaciona con el concepto de strong jump traceability introducido en el capítulo anterior. El teorema fundamental de este capítulo demostrará que los conjuntos c.e. strongly jump traceables son K-triviales. Se demostrará también que existen conjuntos que son K-triviales que no son strongly jump traceable.

Finalmente se hará mención de los últimos resultados en la materia, posibles lineas de investigación y problemas abiertos.

Capítulo 2

Preliminares

En este capítulo repasaremos nociones fundamentales de la teoría de Computabilidad y la teoría algorítmica de la información que serán utilizadas a lo largo de la tesis. Se seguirán principalmente los siguiente libros [32, 24, 17, 1].

2.1. Notación y convenciones

Cadenas. Llamaremos a un elemento de $\{0,1\}$ un *bit*, y a una sucesión finita de bits una *cadena*. Utilizaremos para denotar cadenas letras del alfabeto griego, fundamentalmente las letras que se encuentran en el medio y al final. Denotaremos a las cadenas finitas binarias como $2^{<\omega}$, y a las cadenas binarias de largo n como 2^n, de la misma manera $2^{\leq n}$ denotarán cadenas binarias de largo menor o igual a n. La cadena vacía será λ y el largo de una cadena se denotará como $|\sigma|$, mientras para la concatenación de cadenas σ y τ usaremos según conveniencia $\sigma\tau$ ó $\sigma \frown \tau$. El orden lexicográfico de 2^ω se definirá diciendo que σ es menor a τ (escrito como $\sigma <_L \tau$) si, o bien $|\sigma| < |\tau|$ o $|\sigma| = |\tau|$ y $\sigma(n) = 0$ para el mínimo n tal que $\sigma(n) \neq \tau(n)$. Diremos que σ es un prefijo de τ, notado como $\sigma \preceq \tau$, si $\exists\rho[\sigma\rho = \tau]$. Por otro lado se dirá que σ, τ son *incompatibles*, denotado como $\sigma|\tau$, si no vale ni $\sigma \preceq \tau$ ni $\tau \preceq \sigma$. Sea $str : \mathbb{N} \mapsto 2^{<\omega}$ la enumeración estándar de las cadenas. La cadena $str(n)$ es la secuencia binaria $b_0 b_1 \dots b_m$ para la cual el número binario $1b_0 b_1 \dots b_m$ tiene el valor $n+1$. Con lo cual tenemos que $str(0) = \lambda,\ str(1) = 0,\ str(2) = 1,\ str(3) = 00,\ str(4) = 01$ y así siguiendo. De manera similar notamos $num(\sigma) = n$, con lo cual tenemos, por ejemplo que $num(0^i) = 2^i - 1$ y $num(1^i) = 2^{i+1} - 2$ con lo cual el intervalo $[2^i - 1, 2^{i+1} - 1)$ se encuentra identificado con las cadenas de largo i. Definimos para n entero positivo, $\log n = \text{máx}\{k \in \mathbb{N} : 2^k \leq n\}$. Si $\sigma = str(n)$, luego $|\sigma| = \log(n+1)$. Por ejemplo, si $n = 2^i - 1$, entonces $\sigma = str(n) = 0^i$, y $|\sigma| = \log 2^i = i$.

Conjuntos. Notaremos a los números naturales como $\mathbb{N}$ y al *Espacio de Cantor* de secuencias binarias infinitas como 2^ω. A menos que se especifique lo contrario, cuando hablemos de conjuntos nos estaremos refiriendo a un conjunto de números naturales. Por una *secuencia* entenderemos una secuencia binaria infinita y por un *real* un número real en el intervalo $[0,1]$. En lo que sigue trataremos a estas tres clases de objetos como si

fuesen la misma. Esto es, identificaremos al conjunto A con su función característica χ_A. De esta manera denotaremos $n \in A$ como $A(n) = 1$ y $A(n) = 0$ si $n \notin A$. Esto nos permite identificar A con la secuencia binaria infinita $A(0)A(1)A(2)\ldots$. Si Z es un conjunto y σ es una cadena, entonces podemos escribir $\sigma \preceq Z$, donde vemos a Z como una secuencia binaria, con lo cual nuestra cadena puede verse como un prefijo de Z. Denotaremos como $\overline{A}$ al complemento $\mathbb{N} \setminus A$ de A. Resulta necesario observar que el hecho de que existan reales con más de una representación binaria presenta un problema a la hora de identificar las tres clases de objetos mencionadas con anterioridad. Más precisamente el problema se presenta con los números racionales, sin embargo en estos casos las demostraciones suelen ser directas con lo cual el problema resulta menor. Adoptaremos también la siguiente notación. Para un conjunto A,

$$A \upharpoonright n \text{ denota la cadena } A(0)A(1)\ldots A(n-1)$$

Fijaremos una función de codificación de k-uplas de los números naturales $(n_0, \ldots, n_k)$ al número natural $\langle n_0 \ldots n_k \rangle$. Cuando escribamos $\log n$ nos estaremos refiriendo al logaritmo en base dos de n redondeado al entero más cercano, por convención $\log 0 = 0$

Notación lógica Utilizaremos notación lógica estándar, $\forall^\infty$ querrá decir "para todos excepto un conjunto finito", $\exists^\infty$ querrá decir "existen infinitos" y $\exists^{\geq k}$ denotará "existen por lo menos k". Utilizaremos también la notación usual para λ y μ. $\lambda x f(x, y_0 \ldots, y_n)$ es la función $x \mapsto f(x, y_0, \ldots, y_n)$ para valores fijos $y_0, \ldots, y_n$, $\mu n R(n)$ será el mínimo n para el cual $R(n)$ es válido.

2.2. Conceptos básicos de Computabilidad

En esta sección haremos una revisión de conceptos básicos de la teoría de computabilidad que serán utilizados a lo largo de la presente tesis. En algunos casos esbozaremos la demostración de ciertos teoremas y en otros nos limitaremos a dar las referencias del caso.

Uno de los logros fundamentales de la teoría constituye una definición formal de la noción intuitiva del concepto de función computable. Consideraremos principalmente funciones $f : \mathbb{N}^k \mapsto \mathbb{N}$, aunque bien podríamos tomar cualquier otro objeto finito que pueda codificarse a un número natural. Obtenemos una definición formal de las funciones computables mediante la introducción de las *Máquinas de Turing* [36]. Tales máquinas constan de distintos tipos de cintas: k cintas se utilizarán para los datos de entrada (*input*), otras se utilizarán para procesos internos de la máquina y se reservará una cinta para imprimir los datos de salida (*output*). Las máquinas de Turing poseen un cabezal que utilizarán para leer y escribir datos de un alfabeto finito que incluirá los simbolos 0 y 1. El comportamiento de estas máquinas se describe a través de un número finito de instrucciones, estas instrucciones serán llamadas *Programas de Turing*. En todo momento el programa de Turing debe especificar de manera no ambigua el paso siguiente, contando para ello de la información obtenida de las cintas de entrada y aquella información que se encuentre en las cintas dedicadas a procesos internos. Diremos entonces que una función f es *computable* si existe un programa de Turing P para una máquina con k cintas de entrada, de

manera tal que para toda entrada $x_0, x_1, \ldots, x_{k-1}$ el programa escribe en su cinta de salida el número $f(x_0, x_1, \ldots, x_{k-1})$. Puede suceder que para una entrada particular la máquina no se detenga nunca, de hecho se demuestra que no existe un algoritmo que decida cuándo un programa se va a detener con lo cual resulta natural introducir el concepto de *función parcial*.

Definición 2.2.1. Sea ψ una función cuyo dominio es un subconjunto de $\mathbb{N}^k$ y su rango es un subconjunto de $\mathbb{N}$. Diremos que ψ es una *función parcialmente computable* si existe un programa de Turing P de manera tal que $\psi(x_0, x_1, \ldots, x_{k-1}) = y$ sii P con las entradas $x_0, x_1, \ldots, x_{k-1}$ devuelve en su cinta de salida y. Escribiremos $\psi(x_0, x_1, \ldots, x_{k-1}) \downarrow$ si P se detiene con la entrada $x_0, x_1, \ldots, x_{k-1}$ y $\psi(x_0, x_1, \ldots, x_{k-1}) \uparrow$ en caso contrario. Diremos que ψ es *computable* si ψ es parcialmente computable y $dom(\psi)$ es $\mathbb{N}^k$ donde $dom(\psi) = \{\alpha \in \mathbb{N}^k | \psi(\alpha) \downarrow\}$.

Resulta fundamental el hecho de que las funciones parcialmente computables puedan enumerarse. Sea P_n el n-ésimo programa, entonces denotaremos como Φ_n a la función parcialmente computable dada por el programa P_n. Si $\Phi = \Phi_e$ entonces llamaremos a e un *índice* de Φ. Para funciones parciales introducimos también la siguiente notación: dadas las expresiones α y β

$$\alpha \simeq \beta$$

significará que o bien ambas expresiones están indefinidas, o que están definidas y poseen el mismo valor.

El hecho de que los programas de Turing puedan enumerarse de una manera efectiva nos permite demostrar el siguiente

Teorema 2.2.2. *(Teorema del Parámetro) Para cada función parcialmente computable Θ en dos variables, existe una función q primitiva recursiva y estrictamente creciente tal que*

$$\forall e \forall x \Phi_{q(e)}(x) \simeq \Theta(e, x).$$

Un índice de q puede obtenerse de manera efectiva a partir de un índice de Θ.

El teorema de la recursión constituye una herramienta técnica fundamental de la teoría que puede encontrarse en [11] y dice

Teorema 2.2.3. *Sea $g : \mathbb{N} \mapsto \mathbb{N}$ computable. Existe un e tal que $\Phi_{g(e)} = \Phi_e$. Diremos que e es un punto fijo de g.*

Demostración. Por 2.2.2 existe una función computable q tal que $\Phi_{q(e)}(x) \simeq \Phi_{g(\Phi_e(e))}(x)$ para todo e, x. Sea i tal que $q = \Phi_i$, luego

$$\Phi_{q(i)} = \Phi_{\Phi_i(i)} = \Phi_{g(\Phi_i(i))}.$$

Luego $e = \Phi_i(i) = q(i)$ resulta un punto fijo. □

Definición 2.2.4. Diremos que un conjunto $A \subseteq \mathbb{N}$ es *computablemente enumerable* (c.e. en adelante) si A es el dominio de una función parcialmente computable.

Sea

$$W_e = \mathrm{dom}(\Phi_e).$$

Luego $(W_e)_{e\in\mathbb{N}}$ es una enumeración efectiva de todos los conjuntos c.e. Una sucesión de conjuntos $(S_e)_{e\in\mathbb{N}}$ tal que $\{\langle e, x\rangle : x \in S_e\}$ es c.e. se dirá *uniformemente computable enumerable*.

La función característica f de un conjunto A está dada por $f(x) = 1$ si $x \in A$ y $f(x) = 0$ en caso contrario. Usualmente identificamos A con f. Diremos que A es *computable* si su función característica es computable, si esto no es así entonces diremos que A es *incomputable*.

Teorema 2.2.5. *A es computable $\Leftrightarrow$ A y $A - \mathbb{N}$ son c.e.*

Demostración. $\Rightarrow$: Si A es computable, entonces existe un programa Q_0 que se detiene con la entrada x sii $x \in A$, existe también un programa Q_1 que se detiene en x sii $x \notin A$.

$\Leftarrow$: Fijemos programas Q_0 y Q_1 tales que Q_0 se detiene en x sii $x \in A$, y Q_1 se detiene en x sii $x \notin A$. Para decidir si $x \in A$ corramos los programas Q_0 y Q_1 en paralelo con entrada x hasta que alguno de los dos se detenga. Si Q_0 lo hace primero, la salida es 1 y en caso contrario es 0. □

Podemos obtener un conjunto incomputable $\emptyset'$ mediante un argumento de diagonalización. La idea es definir $\emptyset'$ de manera tal que $\mathbb{N} - \emptyset'$ difiera de W_e para cualquier e. Sea

$$\emptyset' = \{e : e \in W_e\}. \tag{2.1}$$

Llamaremos al conjunto $\emptyset'$ el *halting problem* puesto que $e \in \emptyset'$ únicamente si el programa P_e se detiene con la entrada e.

Proposición 2.2.6. *El conjunto $\emptyset'$ es c.e. pero no es computable.*

Demostración. $\emptyset'$ es c.e. pues $\emptyset' = \mathrm{dom}(J)$ donde J es la función parcial computable dada por $J(e) \simeq \Phi_e(e)$. Si $\emptyset'$ es computable entonces existe un e tal que $\mathbb{N} - \emptyset' = W_e$. Luego $e \in \emptyset' \leftrightarrow e \in W_e \leftrightarrow e \notin \emptyset'$, lo cual constituye un absurdo. □

Corolario 2.2.7. *Si $(A_e)_{e\in\mathbb{N}}$ es una sucesión uniformemente c.e. entonces existe una función computable q tal que $A_e = W_{q(e)}$ para todo e.*

Demostración. Definimos la función parcialmente computable Θ como $\Theta(e, x) \simeq 0$ sii $x \in A_e$ y $\Theta(e, x) \uparrow$ en otro caso. Luego la función obtenida en el teorema 2.2.2 es la requerida. □

Un procedimiento usual dentro de la teoría es la construcción de un objeto, digamos una función o un conjunto c.e. mediante la descripción informal de cada una de sus etapas s. En cada momento se requerirán aproximaciones efectivas de los objetos involucrados en la construcción.

Definición 2.2.8. Escribiremos

$$\Phi_{e,s}(x) = y$$

si $e, x, y < s$ y el programa P_e en la entrada x nos devuelve y en como mucho s pasos de nuestro cómputo. Escribiremos $\Phi_{e,s}(x) \downarrow$ si existe un y tal que $\Phi_{e,s}(x) = y$, y $\Phi_{e,s}(x) \uparrow$ en caso contrario. Más aún, definimos $W_{e,s} = \text{dom}(\Phi_{e,s})$. Observemos que $W_{e,s}$ es finito.

La definición anterior nos dice que en la etapa s poseemos información completa respecto a $\Phi_{e,s}$ y $W_{e,s}$ lo cual es precisamente lo requerido en este tipo de construcciones. Para enunciar lo anterior de una manera formal necesitamos listar de manera efectiva los subconjuntos finitos de $\mathbb{N}$.

Definición 2.2.9. Sea $D_0 =$. Si $n > 0$ tiene la forma $2^{x_1} + 2^{x_2} + \ldots + 2^{x_r}$ donde $x_1 < x_2 < \ldots < x_r$, entonces sea $D_r = \{x_1, \ldots, x_r\}$. Diremos que n es un *índice fuerte* de D_n. Por ejemplo $D_5 = \{0, 2\}$ pues $5 = 2^0 + 2^2$.

Existe una función computable f tal que $f(e, s)$ es un índice fuerte para $W_{e,s}$. Pensamos en una enumeración computable de A como un listado efectivo $a_0, a_1, \ldots$ de elementos de A en algún orden. Formalizaremos esta noción como una unión efectiva de conjuntos finitos (A_s), donde vemos al conjunto A_s como los elementos enumerados en la etapa s. En ciertas etapas podemos elegir no enumerar ningún elemento.

Definición 2.2.10. Una *enumeración computable* de un conjunto A es una sucesión efectiva de índices fuertes $(A_s)_{s\in\mathbb{N}}$ de manera tal que $A_s \subseteq A_{s+1}$ para cada s y además $A = \cup_s A_s$

Cada conjunto W_e posee la aproximación computable $(W_{e,s})_{s\in\mathbb{N}}$. Un *índice* para el conjunto c.e. A es un número e tal que $A = W_e$. Cuando un conjunto A es descripto de esta manera inmediatamente tenemos a nuestra disposición una enumeración computable $(A_s)_{s\in\mathbb{N}}$ de A dada por $A_s = W_{e,s}$.

2.2.1. Reducibilidad de Turing

Una noción fundamental dentro de la teoría de la computabilidad es la de *complejidad computacional relativa*, es decir dados dos conjuntos A y B nos interesa responder a preguntas tales como: "Puede A computarse con la ayuda de B?", "Es A más o menos complejo que B?" En el primer caso diremos que A es computable a partir de B (o *Turing reducible*) si existe un procedimiento computable según el cual dada cualquier entrada n se decide en una cantidad finita de pasos la pertenencia de n al conjunto A. Durante el cómputo nuestro algoritmo tiene permitido hacer consultas al conjunto B, al cual llamaremos *oráculo*, noción que fue introducida por Turing en [37].

Resulta interesante pensar en el oráculo como una extensión que se le realiza a la Máquina de Turing, un agregado de *hardware* que consiste en un nueva cinta infinita que posee la respuesta a la preguntas del tipo "pertenece k a B?" Dentro del plano estrictamente matemático podemos pensar en un oráculo como una suerte de extensión trascendental algebraica.

Las definiciones dadas en 2.2 pueden relativizarse a oráculos en el siguiente sentido, de ahora en más veremos la sucesión efectiva de funciones parcialmente computables $(\Phi_e)_{e\in\mathbb{N}}$ como dependientes de dos argumentos: el oráculo y la entrada usual. Escribiremos $\Phi_e^Y(n)$ en vez de $\Phi_e(Y,n)$ y $\Phi_e^Y(n)\downarrow$ si el programa P_e se detiene con la entrada n cuando se utiliza el oráculo Y, $\Phi_e^Y(n)\uparrow$ significa exactamente lo contrario. Llamaremos *Funcionales de Turing* a Φ_e. Extendiendo definiciones anteriores tambien denotaremos

$$W_e^Y = \mathrm{dom}(\Phi_e^Y).$$

Finalmente damos la siguiente definición formal

Definición 2.2.11. Una función $f:\mathbb{N}\mapsto\mathbb{N}$ se dirá *Turing reducible* a Y, o *computable relativa* a Y, si existe e tal que $f=\Phi_e^Y$. Denotaremos esto como $f\leq_T Y$. Para un conjunto A escribiremos $A\leq_T Y$.

La definición de reducibilidad nos permite definir la siguiente relación de *Equivalencia de Turing*

$$A\equiv_T B \Leftrightarrow A\leq_T B\wedge B\leq_T A.$$

Llamaremos a las clases de equivalencia dadas por $\equiv_T$ *Grados de Turing.*

2.2.2. El Operador de Salto y relativización

Definición 2.2.12. Escribiremos $J^Y(e)\simeq\Phi_e^Y(e)$. El conjunto $Y'=\mathrm{dom}(J^Y)$ se llamará el *salto de Turing* de Y. La función $Y\to Y'$ se llama el operador de salto.

Definición 2.2.13. Definimos $Y^{(n)}$ inductivamente como $Y^{(0)}=Y$ y $Y^{(n+1)}=(Y^{(n)})'$.

Puede verse en Soare [32] que vale $Y<_T Y^{(1)}<_T Y^{(2)}<_T\ldots$

Hecho 2.2.1. *A partir de un funcional de Turing $\Phi=\Phi_e$ se puede obtener de manera efectiva una función primitiva recursiva p estrictamente creciente llamada* función de reducción *para Φ, de manera tal que valga $\forall Y\forall x\Phi^Y(x)=J^Y(\alpha(x))$.*

Demostración. Sea $\Theta^Y(x,y)=\Phi^Y(x)$ (el índice para Θ se obtiene de manera efectiva a partir de e). Puede verse que una versión relativizada del teorema 2.2.2 resulta válida con lo cual existe una función estrictamente creciente p tal que $\forall Y\forall x\Phi^Y_{p(x)}(y)\simeq\Theta^Y(x,y)\simeq\Phi^Y(x)$. Si $y=p(x)$ obtenemos entonces $J^Y(p(x))=\Phi^Y_{p(x)}(p(x))=\Phi^Y(x)$. □

2.2.3. Aproximaciones de funcionales y principio del uso

Definición 2.2.14. Escribiremos $\Phi^Y_{e,s}(x) = y$ si $e, x, y < s$ y el programa P_e en la entrada x nos devuelve y en como mucho s pasos de nuestro cómputo. Por convención en todo momento las consultas a nuestro oráculo serán menores a s también. Escribiremos $\Phi^Y_{e,s}(x)\downarrow$ si existe un y tal que $\Phi_{e,s}(x) = y$, y $\Phi^Y_{e,s}(x)\uparrow$ en caso contrario. Definimos también $W^Y_{e,s} = \text{dom}(\Phi^Y_{e,s})$.

El *principio del uso* nos dice que un cómputo con oráculo que converge sólo realiza finitas consultas al oráculo. Por lo tanto $(\Phi^Y_{e,s})_{s\in\mathbb{N}}$ aproxima Φ^Y_e, es decir

$$\Phi^Y_e(x) = y \leftrightarrow \exists s \Phi^Y_{e,s}(x) = y$$

Definición 2.2.15. El *uso* de $\Phi^Y_e(x)$ se encuentra definido en cuanto que $\Phi^Y_e(x)\downarrow$. En ese caso su valor será de 1+ el valor de la consulta más grande realizada al oráculo y valdrá 1 en caso de que no se haya hecho consulta alguna. Se define de manera similar el valor del uso de $\Phi^Y_{e,s}(x)$, la consulta más grande hecha al oráculo en la etapa s.

Escribiremos

$$\Phi^\sigma_e(x) = y$$

si $\Phi^F_e(x)$ nos devuelve como salida y, donde $F = \{i < |\sigma| : \sigma(i) = 1\}$ y el uso es como mucho $|\sigma|$. Escribiremos $\Phi^\sigma_e(x)\uparrow$ si no existe tal y. Luego para cada conjunto Y,

$$\Phi^Y_e(x) = y \leftrightarrow \Phi^{Y\upharpoonright u}_e(x) = y$$

donde u es el uso de $\Phi^Y_e(x)$. Escribimos $\Phi^\sigma_{e,s}(x) = y$ si $\Phi^\sigma_e(x) = y$ en como mucho s pasos, y $\Phi^\sigma_{e,s}(x)\uparrow$ en caso de que no exista tal y.

Lema 2.2.16. *(Principio del Uso) Sea $\Phi^A(n)$ un cómputo con oráculo convergente, y sea B tal que $B\upharpoonright u = A\upharpoonright u$ donde u es el uso de $\Phi^A(n)$. Luego $\Phi^A(n) = \Phi^B(n)$*

Demostración. Informalmente podemos decir que los conjuntos A y B dan las mismas respuestas a las preguntas relevantes al cómputo, por esta razón el resultado debe ser el mismo. □

2.2.4. Conjuntos Δ^0_2, n-c.e. y ω-c.e

Decir que A es c.e. puede ser pensado como que A posee una aproximación computable que para cada n comienza diciendo $n \notin A$ y luego *cambia de parecer* como mucho una vez, es decir una vez que la aproximación afirma la pertenencia puedo estar seguro de ello. Esta idea se generaliza en [28] donde se pregunta qué sucede si permitimos que nuestra aproximación pueda equivocarse una cantidad finitas de veces, en este caso, si bien la aproximación nos puede dar la respuesta verdadera no podemos estar completamente seguros como en el caso c.e. Resulta interesante por la tanto investigar qué conjuntos surgen si permitimos una cantidad finita de errores y de qué manera difieren éstos de los conjuntos c.e.

Definición 2.2.17. Diremos que un conjunto Z es Δ^0_2 si existe un aproximación computable $(Z_s)_{s\in\mathbb{N}}$ tal que $Z_s \subseteq [0,s]$ y $Z(x) = \lim_s Z_s(x)$.

Resulta de utilidad introducir la siguiente notación. Dada una expresión E que es aproximada en etapas s,

$$E[s]$$

denota el valor de E al *final* de la etapa s. Por ejemplo, dado Z un conjunto Δ^0_2 con una aproximación computable $(Z_s)_{s\in\mathbb{N}}$, escribiremos $\Phi^Z_e(x)[s]$ en vez de $\Phi^{Z_s}_{e,s}(x)$. La gran mayoría de las veces $E[s]$ puede ser evaluada de manera efectiva. Diremos que la expresión E se estabiliza en s si $E[t] = E[s]$ para todo $t \geqslant s$.

En [31] se demuestra que los conjuntos Δ^0_2 coinciden con los conjuntos que son reducibles Turing al *halting problem*.

Lema 2.2.18. *(Límite de Shoenfield) Z es $\Delta^0_2 \Leftrightarrow Z \leq_T \emptyset'$*

Definición 2.2.19.

1. Diremos que Z es ω-c.e. si Z es Δ^0_2 y existe una función computable b tal que

$$b(x) \geq \#\{s > x : Z_s(x) \neq Z_{s-1}(x)\} \text{ para cada } x.$$

2. Si $Z_s(s-1) = 0$ para cada $s > 0$ y $b(x)$ se puede elegir de valor constante n, entonces diremos que Z es n-c.e.

Luego Z es 1-c.e. sii Z es c.e., y Z es 2-c.e. sii $Z = A - B$ para conjuntos c.e. A, B, llamaremos a estos conjuntos también diferencia de c.e. (d.c.e) Puede verse en general que A es un conjunto $2n+2$-c.e. sii A es la unión de $n+1$ conjuntos d.c.e, y A es $2n+1$-c.e. sii es la unión de n conjuntos d.c.e y un conjunto c.e.

Tenemos por lo tanto la siguiente jerarquía:

$$\text{computable} \subset \text{c.e.} \subset \text{2-c.e.} \subset \text{3-c.e.} \subset \ldots \subset \omega\text{-c.e.}$$

Esta jerarquía resulta propia incluso si se consideran grados de Turing. En el caso de los n-c.e. la demostración de que son conjuntos propios puede obtenerse mediante un argumento de diagonalización donde se usa, como es de esperar, el hecho de que la aproximación puede poseer un cambio más.

2.2.5. Árboles, caminos y clases Π^0_1 y Σ^0_1

Un *árbol* es un subconjunto de $2^{<\omega}$ cerrado bajo prefijos iniciales. Es decir, si T es un árbol $x \in T$ y $y \preceq x$ entonces $y \in T$. Un *camino* a lo largo del árbol T es una secuencia infinita $P \in 2^\omega$ tal que si $\sigma \prec P$ entonces $\sigma \in T$. Denotaremos al conjunto de caminos de un árbol T como $[T]$. Podemos visualizar un árbol como creciendo hacia arriba con $\sigma 0$ a la izquierda de $\sigma 1$, de esta manera podemos utilizar una terminología visual que resulta útil en muchos casos, hablando de izquierda, derecha y arriba.

Por ejemplo $T = \{0^i : i \in \mathbb{N}\} \cup \{0^i 1 : i \in \mathbb{N}\}$ es un árbol tal que $[T] = \{0^\infty\}$.

Lema 2.2.20. *(König) Si B es un árbol infinito entonces $[B] \neq \emptyset$.*

Demostración. Para cada n sea x_n la cadena más a la izquierda de largo n tal que $B \cap \{y : y \succeq x\}$ es infinito. Luego $x_n \preceq x_{n+1}$ para cada n, y $\cup_n x_n$ es un camino de B que extiende a x. □

Un subconjunto de 2^ω se llamará una clase Π^0_1 si es igual a $[T]$ para algún árbol computable T. Una formulación equivalente a la de ser una clase Π^0_1 es considerar una relación computable R, entonces diremos que C es una clase Π^0_1 si

$$C = \{\alpha \in 2^\omega : \forall n R(\alpha \upharpoonright n)\}.$$

No resulta difícil demostrar que ambas definiciones resultan equivalentes.

El complemento de una clase Π^0_1 es una clase Σ^0_1. Por lo tanto C es una *clase* Σ^0_1 si

$$C = \{\alpha \in 2^\omega : \exists n R(\alpha \upharpoonright n)\}.$$

Hecho 2.2.2. *Sea C una clase Σ^0_1. Entonces existe un conjunto c.e. $W \subseteq 2^{<\omega}$ tal que $C = \cup\{[\sigma] : \sigma \in W\}$ donde $[\sigma] := \{Z : \sigma \preceq Z\}$.*

El siguiente resultado pertenece al folklore de la teoría

Proposición 2.2.21. *Sea C una clase Π^0_1 con una cantidad finita de miembros. Luego cada miembro de C es computable.*

Demostración. Sea T un árbol computable tal que $C = [T]$. Entonces T tiene finitos caminos. Sea P un camino de T, debe haber un $\sigma \preceq P$ tal que ningún otro camino a través de T extiende a σ. Luego para todo $n > |\sigma|$, existe exactamente un $\tau \succ \sigma$ tal que la parte de T arriba de τ es infinita (aquí utilizamos el Lema 2.2.20). Luego para computar $P \upharpoonright n$ para $n > |\sigma|$ buscamos un $m \geq n$ de manera que exactamente un $\tau \succ \sigma$ de largo n tenga una extensión de largo m en T. Luego $P \upharpoonright n = \tau$. □

2.3. Complejidad de Kolmogorov

Esta sección es una breve introducción a la teoría de complejidad de Kolmogorov y a la teoría algorítmica de cadenas finitas aleatorias. En [17] puede encontrarse una discusión detallada de los resultados principales así como también las raíces históricas y filosóficas de la teoría. En lo que sigue trataremos básicamente con dos tipos de complejidades de Kolmogorov: complejidad plana y complejidad libre de prefijos. A la primera la notaremos con la letra C y a la segunda con la letra K. Se verán los beneficios y las desventajas de cada una de estas complejidades.

2.3.1. Complejidad Simple de Kolmogorov

La idea principal de la complejidad simple de Kolmogorov se sigue de la siguiente observación: Consideremos las siguiente cadenas de 24 bits

101010101010101010101010

101011011100011110110011

Si bien ambas cadenas son del mismo largo y tienen por lo tanto la misma probabilidad de aparecer, si se considera una probabilidad uniforme, existe una marcada diferencia entre ambas: nos resulta más sencillo *describir* la primera que la segunda. La primer cadena puede ser descripta como un una sucesión de 24 bits, con un 1 en la posición n sii n par. La tarea no es tan sencilla en el segundo caso, y si se nos pide una descripción no podemos menos que recitar la cadena tal cual es. Esto es, el patrón hallado en la primera cadena nos permite "comprimir" la descripción, mientras que en el segundo caso una descripción concisa no resulta fácil de obtener a simple vista. Esa discusión ha sido llevada dentro del terreno de lo informal puesto que en ningún momento se aclaró a qué nos referimos con una descripción de una cadena. Sea $f : 2^{<\omega} \mapsto 2^{<\omega}$. Diremos que τ es una f-descripción de σ si $f(\tau) = \sigma$. Si bien esta definición es más rigurosa si no se tiene cuidado o se la hace más restrictiva pueden surgir todo tipo de paradojas.

Un ejemplo de ello es la llamada *paradoja de Berry* discutida por Bertrand Russell en [29], la cual surge de considerar la siguiente expresión: *El mínimo entero positivo no describible en menos de doce palabras.* Como la cantidad de palabras es finita existe entonces una cantidad finita de frases de menos de doce palabras. Por el otro lado como los enteros son infinitos existen números que no tendrán una descripción en menos de doce palabras, debe existir por lo tanto un mínimo número que cumpla la propiedad mencionada. A este número se refiere la expresión. Notemos sin embargo que la expresión arriba mencionada posee *once* palabras con lo cual nuestro número posee una descripción en menos de doce palabras, resultando este último hecho claramente paradójico. La paradoja surge debido a que no se especifica de manera precisa qué significa ser *describible*.

Con el objeto de no caer en ninguna paradoja pediremos que $f : 2^{<\omega} \mapsto 2^{<\omega}$ sea parcialmente computable. En lo que sigue, llamaremos a este tipo de funciones *máquinas*. La formalización de estos conceptos fue llevada a cabo en parte por Solomonoff en [33] e independientemente por Kolmogorov en [12].

Sea M una máquina. La *complejidad de Kolmogorov* de una cadena σ con respecto a M es

$$C_M(\sigma) := \text{mín}\{|\tau| : M(\tau) = \sigma\},$$

donde este mínimo se toma como ∞ si el conjunto es vacío. Diremos que σ es aleatorio relativo a M si $C_M(\sigma) \geq |\sigma|$. En este caso pensaremos a M como un "sistema de descripción" de nuestra cadena, es decir, relativo a nuestro sistema de descripción diremos que σ es M-aleatorio si no posee una descripción corta. Querríamos poder definir un concepto

de aleatoriedad que no dependa de una máquina en particular debido a que podría ser que con una máquina nuestra cadena posea una descripción corta y con otra no. Si nos permitimos cierta laxitud podemos ver cómo es posible esto último en el siguiente ejemplo: Consideremos la siguiente expansión en fracciones continuas del número de oro

$$[1,1,1,1,\ldots] = 1 + \cfrac{1}{1 + \cfrac{1}{+\cfrac{1}{1+\ldots}}} \tag{2.2}$$

Como puede verse, la representación de éste número en fracciones continuas resulta particularmente sencilla, y el patrón resulta claro, sin embargo de su representación decimal 1,6180339887... no emerge un patrón con similar claridad. Querríamos por lo tanto hallar un sistema de descripción *universal*. Este sistema de descripción tendría la capacidad de simular cualquier otro sistema con un incremento constante en la longitud de las descripciones. Un avance fundamental de la teoría es ver que este sistema de descripción universal existe si se consideran ciertas restricciones sobre el sistema de representación. Más precisamente buscamos una máquina U que cumpla para todo σ

$$C_U(\sigma) \leq C_M(\sigma) + e_M$$

para un e_M constante que llamaremos *constante de codificación* de M. Una tal máquina U existe debido a que existe una enumeración efectiva de todas las funciones parcialmente computables $(\Phi)_{e\in\mathbb{N}}$. Esto último nos permite dar la siguiente

Definición 2.3.1. La *máquina universal standard* $\mathbb{V}$ está dada por

$$\mathbb{V}(0^{e-1}1\rho) \simeq \Phi_e(\rho) \tag{2.3}$$

para cada $e > 0$ y $\rho \in \{0,1\}^*$.

Resulta luego sensato definir

$$C(\sigma) := C_U(\sigma)$$

Resulta claro que una máquina universal distinta daría orígen a una definición distinta de C, sin embargo ambas definiciones diferirían únicamente en una constante. Denotaremos por $C(\sigma,\tau)$ al número $C(\langle\sigma,\tau\rangle)$. Supongamos que $C(\sigma) = k$. Entonces existe una cadena τ de largo k tal que $U(\tau)$ converge con valor σ. Fijamos una cadena τ mínima en el orden lexicográfico que cumple lo anterior y la denotaremos como σ^*_C.

Los siguiente resultados poseen una demostración directa:

Lema 2.3.2. *(Kolmogorov[12])*

1. $C(\sigma) \leq |\sigma| + O(1)$.

2. $C(\sigma\sigma) \leq C(\sigma) + O(1)$.

3. Si $h : 2^{<\omega} \mapsto 2^{<\omega}$ *es una función computable. Entonces* $C(h(\sigma)) \leq C(\sigma) + O(1)$.

Por ejemplo podemos elegir $f : 2^{<\omega} \mapsto 2^{<\omega}$ la función identidad, luego tenemos $C(\sigma) \leq C_f(\sigma) + O(1) = |\sigma| + O(1)$. Los otros resultados se demuestran de manera similar.

Diremos que una cadena es C-aleatoria si es aleatoria relativa a U, es decir si $C(\sigma) \geq |\sigma|$ El siguiente resultado nos dice que tales cadenas existen.

Teorema 2.3.3. *(Solomonoff [33], Kolmogorov [12]) Para cada* n *existe una cadena* σ *con* $|\sigma| = n$ *y* $C(\sigma) \geq n$.

Demostración. Existen $2^n - 1$ cadenas binarias de largo menos de n, con lo cual hay como mucho $2^n - 1$ cadenas binarias de complejidad menor a n. □

Notemos que con un argumento combinatorio similar obtenemos que para todo k y para todo n,

$$|\{\sigma \in 2^n : C(\sigma) > |\sigma| - k\}| > 2^n(1 - 2^{-k})$$

Desearíamos que valga la siguiente propiedad $C(\sigma, \tau) \leq C(\sigma) + C(\tau) + O(1)$, pero no es el caso. El problema fundamental es que no podemos concatenar descripciones de σ y de τ debido a que nos resulta imposible reconocer cuando termina una descripción y cuando comienza la otra. De hecho, existen cadenas suficientemente largas cuyos prefijos iniciales son altamente compresibles.

Lema 2.3.4. *(Martin-Löf [19]) Sea* k *fijo.Si* μ *es suficientemente largo entonces existe un prefijo inicial* σ *de* μ *tal que* $C(\sigma) < |\sigma| - k$. *Luego para cualquier* d *fijo, tenemos* $\mu = \sigma\tau$ *tal que* $C(\mu) > C(\sigma) + C(\tau) + d$.

Demostración. Sea ν un segmento inicial de μ y sea n tal que ν es la n-ésima cadena en el orden lexicográfico de cadenas. Sea ρ la cadena consistente en los próximos n bits de μ luego de ν, y sea $\sigma = \nu\rho$. Para generar σ basta conocer ρ, dado que una vez que contamos con esta cadena podemos obtener su longitud, que será n y luego podemos buscar la n-ésima cadena en el orden lexicográfico, luego concatenamos ambos cadenas y obtenemos σ, en conclusión tenemos que $C(\sigma) \leq |\rho| + c$, donde c no depende de la elección de ρ sino de la máquina universal U seleccionada. Como μ es suficientemente grande podemos pedir que $|\nu| > c + k$ y como $|\sigma| = |\nu| + |\rho|$ entonces obtenemos $C(\sigma) < |\sigma| - k$.

Para la segunda parte del lema, sea c tal que $C(\tau) < |\tau| + c$ para todo τ y sea $k = c + d$. Sea μ una cadena suficientemente larga tal que $C(\mu) \geq |\mu|$. Por la primer parte del lema podemos escribir $\mu = \sigma\tau$ tal que $C(\sigma) < |\sigma| - k$. Entonces $C(\mu) \geq |\mu| = |\sigma| + |\tau| > C(\sigma) + k + C(\tau) - c = C(\sigma) + C(\tau) + d$

□

Por el otro lado, con técnicas similares a las del lema 2.3.4 obtenemos la siguiente cota que puede verse que es bastante precisa.

$$C(\sigma\tau) < C(\sigma) + C(\tau) + 2\log|\sigma| + O(1).$$

Resulta útil en muchos casos medir la compresibilidad de una cadena y dada una cadena x. Para ello consideremos $C(y|x)$ como el largo de la menor descripción de y con x como información auxiliar. Para formalizar esta noción introducimos máquinas de Turing que admiten dos entradas, llamadas *máquinas binarias*. Podemos considerar de manera similar a las máquinas de una entrada una máquina binaria universal dada por

$$\mathbb{V}^2(0^{e-1}1\sigma, y) \simeq \Phi_e^2(\sigma, y)$$

donde $e > 0$ y $\sigma \in \{0,1\}^*$. Definimos entonces la *complejidad condicional de Kolmogorov* de y dado x como

$$C(y|x) = \text{mín}\{|\sigma| : \mathbb{V}^2(\sigma, x) = y\}.$$

2.3.2. Máquinas libres de prefijos y complejidad

En esta subsección introduciremos una variante de la complejidad plana de Kolmogorov que nos permitirá recuperar una propiedad deseable como la subaditividad de la complejidad de dos cadenas concatenadas. En la subsección anterior estudiamos la complejidad plana de Kolmogorov C la cual resulta a nivel conceptual bastante intuitiva, sin embargo esta simplicidad trae aparejada ciertos incovenientes, uno de los cuales ya fue mencionado en el Lema 2.3.4. Una de las raíces de estos incovenientes proviene del hecho de que nuestra definición de C no impone (más allá de pedir que la máquina sea parcialmente computable) ninguna otra restricción.

Un camino natural a seguir a la hora de subsanar los incovenientes de la complejidad simple consiste en restringir el conjunto de máquinas consideradas. En esta subsección tal restricción operará sobre el conjunto de *Máquinas libres de prefijos*. Una máquina M se dirá libre de prefijos si su dominio es un conjunto libre de prefijos, esto es $\forall\sigma, \rho \in \text{dom } M[\sigma \preceq \rho \to \sigma = \rho]$. Con el objeto de indicar que la máquina M es libre de prefijos, escribiremos $K_M(x)$ en vez de $C_M(x)$. Se desarrollará una teoría paralela a la complejidad plana de Kolmogorov basada en este tipo de máquinas. Dada una cadena x la complejidad libre de prefijos de x será denotada como $K(x)$. Veremos que es ésta última noción la que resulta de mayor utilidad a la hora de estudiar la interacción entre computabilidad y aleatoriedad, sin dejar de mencionar la importancia de C en ciertas ocasiones.

En lo que sigue haremos mención de las propiedades básicas de la complejidad libre de prefijos K. De manera similar al caso de complejidad plana de Kolmogorov podemos dar la siguiente

Definición 2.3.5. Diremos que una máquina R libre de prefijos es *universal*, si para cada máquina M libre de prefijos existe una constante d_M tal que $\forall x K_R(x) \leq K_M(x) + d_M$.

Como antes, la constante d_M se llamará la constante de codificación de M con respecto a R. Puede verse que existe una manera efectiva de listar las máquinas libres de prefijos $(M_d)_{d\in\mathbb{N}}$, lo cual nos permite demostrar la siguiente

Proposición 2.3.6. *Existe una máquina universal* $\mathbb{U}$ *libre de prefijos definida como* $\mathbb{U}(0^{d-1}1\sigma) \simeq M_d(\sigma)$ *para* $d > 0$.

Definimos luego la *complejidad libre de prefijos* de una cadena σ como

$$K(x) := K_{\mathbb{U}}(x) = \text{mín}\{|\sigma| : \mathbb{U}(\sigma) = x\}.$$

Como en el caso de la complejidad plana de Kolmogorov la elección de la máquina universal no afecta la definición de K a menos de una constante aditiva. Con la definición anterior logramos la subaditividad de la cual carecíamos en la complejidad simple.

Hecho 2.3.1. *Si* $K(x, y) = K(\langle x, y\rangle)$ *entonces tenemos que* $K(x, y) \leq K(x)+K(y)+O(1)$.

Demostración. Notemos que en esta caso nos hemos librado del término logarítmico presente en el caso de complejidad simple. Podemos lograr la desigualdad debido al hecho que al considerar programas libres de prefijos podemos concatenar descripciones sin nececidad de indicar cuando una descripción termina y cuando comienza otra. De manera más precisa, sea $\mathbb{U}(x^*) = x$ y con $|x^*| = K(x)$ y $\mathbb{U}(y^*) = y$ con $|y^*| = K(y)$. Como los programas de $\mathbb{U}$ son libre de prefijos podemos obtener una máquina V que primero lee x^* y computa x y luego lee y^* y computa y para finalmente computar $\langle x, y\rangle$. □

El siguiente lema, similar al caso de complejidad plana, es sencillo aunque importante.

Lema 2.3.7. *Si* $h : 2^{<\omega} \mapsto 2^{<\omega}$ *es una función computable entonces* $K(h(\sigma)) \leq K(\sigma) + O(1)$.

En el caso C obteníamos una cota computable $C(x) \leq |x| + O(1)$ bastante acertada. Una de las desventajas de K es que no existe una "buena" cota computable en términos del largo de x. La cota para C se obtenía a través del uso de la máquina Φ_1 que copia su entrada, es decir $\Phi_1(x) = x$, el problema con ésta máquina es que no es libre de prefijos, de hecho puede verse que la cota anterior para C falla en ciertas ocasiones. Si queremos obtener rápidamente una cota computable de K sin importarnos qué tan buena es, podemos considerar la máquina M definida como $M(0^{|x|}1x) = x$ para cada cadena x. Esta máquina resulta claramente libre de prefijos puesto que $0^{|x|}1x \preceq 0^{|y|}1y$ implica primero que $|x| = |y|$ y luego que $x = y$. Obtenemos entonces que

$$K(x) \leq 2|x| + O(1) \tag{2.4}$$

La idea básica de la máquina es anteponer a la cadena x una descripción de su largo $n = |x|$, cualquier otra forma de codificar su largo habría servido. En este caso nos preguntamos si podemos lograr una codificación más sucinta con el objeto de mejorar la cota.

Proposición 2.3.8. $K(x) \leq K(|x|) + |x| + O(1) \leq 2\log|x| + |x| + O(1)$ *para cada cadena* x.

Demostración. La segunda desigualdad se sigue de aplicar (2.4) a $|x|$. Para la primera desigualdad definimos la siguiente máquina N: con la entrada τ buscamos una descomposición $\tau = \sigma x$ y $U(\sigma)\downarrow = |\sigma| = n$ y si un tal σ es hallado devolver x. Claramente N es libre de prefijos. Dado x sea σ la U-descripción más corta de $|x|$. Entonces $U(\sigma x) = x$ esto nos muestra que $K(x) \leq K(|x|) + |x| + O(1)$. □

Si bien no contamos con una buena cota computable para K tenemos por el otro lado una aproximación desde abajo que es computable. Específicamente tenemos

$$K_s(\sigma) = \text{mín}\{|\tau| : U(\tau)[s]\downarrow = \sigma\}$$

En otras palabras $K_s(\sigma)$ es el mínimo programa que describe σ dado por U en menos de s pasos. Desde luego puede que tal programa no exista. Como resulta útil que K_s esté siempre definida, definiremos en los restantes casos a $K_s(\sigma)$ como $2|\sigma|+c$, esto puede hacerse debido a que existe c tal que $K(\sigma) \leq 2|\sigma|+c$ para todo σ. Tenemos luego las siguientes propiedads de K_s

1. La función $(s, \sigma) \mapsto K_s(\sigma)$ es computable,
2. $K_s(\sigma) \geq K_{s+1}(\sigma)$ para todo s y σ y,
3. $K(\sigma) = \lim_s K_s(\sigma)$ para todo σ.

El teorema de Kraft-Chaitin

Una de las herramientas principales a la hora de construir máquinas libres de prefijos es una efectivización de una desigualdad de Kraft [13]. Puede verse que si $A \subset 2^{<\omega}$ es libre de prefijos entonces $\sum_{\sigma\in A} 2^{-|\sigma|} \leq 1$. De manera recíproca supongamos que tenemos una sucesión $\{d_i\}$ de números natural tales que $\sum_i 2^{-d_i} \leq 1$. Como en este caso no nos interesa argumentar de una manera efectiva podemos asumir que $d_0 \leq d_1 \leq \ldots$ reorganizando los d_i en caso de ser necesario. Sea σ_i la primera cadena de largo d_i en el orden lexicográfico que no extiende a σ_j para ningún $j < i$. Una inducción sencilla nos muestra que una tal cadena existe siempre. Vemos por lo tanto que para cualquier sucesión d_i que cumpla $\sum_{\sigma\in A} 2^{-|\sigma|} \leq 1$ existe un un sucesión libre de prefijos σ_i tal que $|\sigma_i| = d_i$ para todo i. El teorema de Kraft Chaitin es una efectivización de este hecho.

Teorema 2.3.9. *(Levin [15], Schnorr [30], Chaitin [4]) Sea* $\langle d_i, \tau_i\rangle_{i\in\mathbb{N}}$ *una secuencia computable de pares (que llamaremos* peticiones*) tales que* $d_i \in \mathbb{N}$ *y* $t_i \in 2^{<\omega}$ *tales que* $\sum_i 2^{-d_i} \leq 1$. *Luego existe una máquina* M *libre de prefijos y cadenas* σ_i *de largo* d_i *tales que* $M(\sigma_i) = \tau_i$ *para todo* i *. Más aún, un índice para* M *puede obtenerse de manera efectiva a partir de un índice de nuestra sucesión de peticiones.*

Veamos una aplicación sencilla del teorema de Kraft-Chaitin. Sea $\tau_0, \tau_1, \ldots$ una enumeración efectiva de $2^{<\omega}$ con τ_0 la cadena vacía. Sea $d_0 = 2$ y para $d > 0$ sea $d_i = |\tau_i| + 2\log|\tau_i| + 2$. Entonces

$$\sum_i 2^{-d_i} \leq \frac{1}{4} + \sum_{i>0} \frac{2^{-|\tau_i|}}{2|\tau_i|^2} = \frac{1}{4} + \sum_{n>0} 2^n \frac{2^{-n}}{2n^2} = \frac{1}{4} + \frac{1}{2}\sum_{n>0} \frac{1}{n^2} < 1$$

Por lo tanto $\{\langle d_i, \tau_i\rangle\}_{i\in\mathbb{N}}$ es un conjunto de Kraft-Chaitin, y obtenemos la siguiente cota superior

$$K(\tau) \leq |\tau| + 2\log|\tau| + O(1)$$

Veremos que la cota superior dada con anterioridad no puede ser mejorada en la gran mayoría de los casos. Notemos que como $K = K_U$ y U es libre de prefijos entonces tenemos que $\sum_{\sigma\in 2^{<\omega}} 2^{-K(\sigma)} \leq \sum_{U(\tau)\downarrow} 2^{-|\tau|} \leq 1$.

Lema 2.3.10. *Para cualquier d, existe un σ tal que $K(\sigma) > |\sigma| + \log|\sigma| + d$.*

Demostración. Supongamos que $K(\sigma) \leq |\sigma| + \log|\sigma| + d$ para todo σ. Entonces

$$\sum_\sigma 2^{-K(\sigma)} \geq \sum_{|\sigma|>0} \frac{2^{-|\sigma|-d}}{|\sigma|} = \sum_{n>0} 2^n \frac{2^{-n-d}}{n} = 2^{-d}\sum_{n>0} \frac{1}{n} = \infty$$

lo cual es un absurdo. □

De hecho sin mayor modificación podems probar el siguiente

Corolario 2.3.11. *Sea $f : 2^{<\omega} \mapsto \mathbb{N}$ tal que $\sum_{\sigma\in 2^{<\omega}} 2^{-f(\sigma)} = \infty$ entonces $K(\sigma) > |\sigma| + f(\sigma)$.*

2.4. Aleatoriedad de los reales

En esta sección introduciremos una noción matemática que se corresponderá con nuestra noción intuitiva de lo que debe ser un conjunto o real aleatorio.

La definición de un real aleatorio puede abordarse básicamente desde tres ángulos distintos

1. *El enfoque del programador* Un real aleatorio es aquel cuyos prefijos iniciales resultan difíciles de comprimir a través de un programa.

2. *El enfoque estadístico* Los reales aleatorios son aquellos números que no poseen *propiedades efectivas raras* o dicho de otro modo son los reales que pasan todos los tests estadísticos efectivos.

3. *El enfoque del apostador* Este enfoque probablemente sea el más natural desde el punto de vista matemático. Diremos que una sucesión de bits es aleatoria si nos resulta imposible predecir el próximo bit incluso si contamos con la información de todos los bits anteriores.

En esta sección trataremos únicamente los dos primeros enfoques puesto que éstos serán los que utilizaremos a lo largo de la tesis, para una desarrollo pormenorizado del tercer enfoque puede consultarse [17].

2.4.1. El enfoque del programador

Un primer intento de definición de real aleatorio podría ser: A es aleatorio si $C(A \upharpoonright n) \geq n - O(1)$. Desafortunadamente puede verse que ningún real cumple esta condición. Esta observación fundamental pertence a Martin-Löf y se sigue del Lema 2.3.4, así como también la observación de que la desigualdad $C(xy) \leq C(x) + C(y) + O(1)$ tampoco vale. Nuestra intuición sin embargo es correcta, basta reemplazar la complejidad plana de Kolmogorov por la complejidad libre de prefijos. De esta forma arribamos a la siguiente

Definición 2.4.1. (Levin [16], Chaitin [3]) Un real A es 1-aleatorio si $\exists c \; \forall n \; K(A \upharpoonright n) \geq n - c$.

Acaso el ejemplo más famoso de real aleatorio sea el siguiente

Teorema 2.4.2. *(Ω de Chaitin [3]) La siguiente probabilidad Ω resulta un real 1-aleatorio*

$$\Omega = \sum_{\mathbb{U}(\sigma)\downarrow} 2^{-|\sigma|}$$

La suma anterior puede verse como el resultado del siguiente experimento debido a Chatin: consideremos la máquina libre de prefijos M, es decir su dominio es un conjunto libre de prefijos, cada vez que M requiere la entrada de un bit decidimos si es 0 o 1 lanzando una moneda. Si el experimento continua de manera indefinida habremos obtenido una sucesión infinita de ceros y unos, nos preguntamos entonces si algún prefijo de esta sucesión infinita pertenece al dominio de M, o dicho de otra forma, si el programa que constituye el prefijo de la sucesión eventualmente se detiene. Que la suma sea una probabilidad se sigue del teorema de Kraft y del hecho de que el dominio de M es un conjunto libre de prefijos.

Resulta interesante observar, sin embargo, que puede utilizarse la complejidad plana de Kolmogorov para caracterizar la 1-aleatoriedad. Dicha caracterización pertence a Miller y Yu [20] y dice

Teorema 2.4.3. *Son equivalentes*

1. *A es 1-aleatorio.*
2. *$\forall n C(A \upharpoonright n) \geq n - g(n) - c$ para toda función computable g tal que $\sum_n 2^{-g(n)} < \infty$.*

2.4.2. El enfoque estadístico

La idea fundamental consiste en considerar como aleatorios a aquellos reales que no poseen propiedades raras. Uno de los primeros intentos de formalizar la noción de aleatoriedad se debe a von Mises [39] quién argumentó que un real aletorio es aquel que pasa todos los tests estadísticos. Por ejemplo, si α es un real aleatorio, entonces esperaríamos que se cumpla la ley de los grandes números, es decir $\lim_{n\to\infty} \frac{\alpha(1)+\ldots+\alpha(n)}{n} = \frac{1}{2}$. Si denotamos como L la clase de reales que satisface la ley de los grandes números entonces $\mu(\bar{L}) = 0$, donde μ es la medida de Lebesgue. Es decir esta clase es nula. Nuestra intuición nos dice entonces que las clases nulas, poseedoras de propiedades extraordinarias, no deben contener ningún real aleatorio. Si bien la intuición de von Mises era correcta su noción de tests estadísticos resultó poco precisa y, como posteriormente demostró Ville [38], implicaba incluso la inexistencia de tales reales aleatorios.

Debemos a Martin-Löf una noción precisa de *test estadístico* que nos permite definir la aleatoriedad de un real de manera adecuada. Martin-Löf logró tal definición efectivizando la noción de conjunto de medida nula.

Definición 2.4.4. 1. (Martin-Löf [19]) Un *test* de Martin-Löf es una sucesión $\{U_n\}_{n\in\mathbb{N}}$ de clases Σ^0_1 uniformes tales que $\mu(U_n) \leq 2^{-n}$ para todo n.

2. Una clase $C \subset 2^\omega$ es Martin-Löf *nula* si existe un test de Martin-Löf $\{U_n\}_{n\in\mathbb{N}}$ tal que $C \subset \cap_n U_n$.

3. Un conjunto $A \in 2^\omega$ es Martin-Löf aletorio si $\{A\}$ no es Martin-Löf nulo.

El siguiente teorema fundamental justifica la solidez de ambos enfoques demostrando que son esencialmente equivalentes.

Teorema 2.4.5. *(Schnorr [30]) Un conjunto es Martin-Löf aletorio si y sólo si es* 1*-aleatorio.*

Una propiedad importante de los test de Martin-Löf es la existencia de un test *universal* en el siguiente sentido

Teorema 2.4.6. *(Martin-Löf [19])Diremos que un test de Martin-Löf* $\{U_n\}_{n\in\mathbb{N}}$ *es* universal *si* $\cup_n U_n$ *contiene cada conjunto Martin-Löf nulo. Es decir, para cualquier test* $\{V_n\}_{n\in\mathbb{N}}$ *tenemos que* $\cap_n V_n \subseteq \cap_n U_n$*. Existe un test universal de Martin-Löf.*

En otras palabras esto nos dice que no hace falta utilizar todos los tests sino que basta uno solo. Un ejemplo de test de Martin-Löf es el siguiente: Sea $U_k = \{X : \exists n K(X \upharpoonright n) \leq n - k\}$ puede demostrarse que $\{U_k\}_{k\in\mathbb{N}}$ es un test universal.

Capítulo 3

Jump Traceability

Informalmente una función f de $\mathbb{N} \mapsto \mathbb{N}$ es *trazable* si podemos obtener de una manera sencilla una colección finita de posibles valores para cada $f(x)$. Llamaremos a esta colección de posibles valores una *traza* de f, generalmente se pide que la traza sea computable o c.e.

Recordemos que una sucesión de conjuntos $(T_e)_{e\in\mathbb{N}}$ tal que $\{\langle e, x\rangle : x \in T_e\}$ es c.e. se dirá *uniformemente computable enumerable*.

Definición 3.0.7. 1. Una família uniformemente c.e. $\{T_e\}_{e\in\mathbb{N}}$ de conjuntos finitos es una *traza c.e* si para una función computable h, para todo e $|T_e| \leq h(e)$. Diremos que h es una cota de T.

2. El conjunto A es *jump-traceable* si existe una traza c.e $\{T_e\}_{e\in\mathbb{N}}$ tal que

$$\forall e\ J^A(e)\downarrow \Rightarrow J^A(e) \in T_e.$$

Debido a la universalidad del salto, la definición anterior no sólo restringe los posibles valores del salto sino también los valores de cualquier función A-computable.

Hecho 3.0.1. *Si A es jump-traceable vía $\{T_e\}_{e\in\mathbb{N}}$, entonces existe una traza $\{S_e\}_{e\in\mathbb{N}}$ tal que para cada funcional Ψ,*

$$\forall^\infty[\Psi^A(m)\downarrow \Rightarrow \Psi^A(m) \in S_m]$$

Definamos $\{S_e\}_{e\in\mathbb{N}}$ como $S_m = \cup_{i\leq m} T_{\alpha_i(m)}$, donde $\alpha_i(m)$ es una enumeración de todas las funciones primitivas recursivas de aridad 1.

Para enunciar los teoremas de la próxima sección debemos introducir también la siguiente

Definición 3.0.8. A es *superlow* sii A' es un conjunto ω-c.e.

Notemos que las propiedades de ser *jump traceable* y *superlow* son bastante distintas: mientras la propiedad de ser jump traceable expresa que la función J^A tiene un rango pequeño, la propiedad de ser superlow restringe la complejidad computacional del dominio $A' = \{e : J^A(e)\downarrow\}$. Sin embargo Nies [25] demostró que en el caso c.e son equivalentes, es decir

Teorema 3.0.9. *Sea A c.e. Entonces A es jump traceble $\Leftrightarrow$ A es superlow.*

3.1. Conjuntos superlow y jump-traceables

Los resultados de las siguientes subsecciones pueden hallarse en la tesis doctoral de Keng Meng Ng [22], el caso para $n = 1$ se debe a Nies [25].

3.1.1. Todo conjunto n-c.e. y jump traceable es superlow

Un ingrediente fundamental de la demostración de Nies [25] es el uso que hace de la aproximación de A. Siendo A c.e. a medida que nuestra aproximación avanza podemos descartar las aproximaciones anteriores y quedarnos con la última versión de la aproximación con la seguridad de que cómputos anteriores no volverán a aparecer una vez que nos hemos alejado de ellos dado que la cantidad de cambios por bit se encuentra limitada a uno. Sin embargo, si nuestro conjunto no es c.e. no puede decirse lo mismo. Como nuestra aproximación para un bit específico puede cambiar de 0 a 1 más de una vez no tenemos la certeza de contar con la "mejor" versión en la última etapa. Esta dificultad puede eludirse en el caso en que A es n-c.e. Esto es posible puesto que los cambios de parecer de nuestra aproximación están acotados de una manera constante lo cual nos permitirá, siendo más cuidadosos, llegar al mismo resultado.

Resulta útil ver las construcciones efectuadas en las siguientes demostraciones como un juego entre nosotros y un *oponente*, nuestro objetivo, por lo general, es la construcción de un objeto, sea este un conjunto o una función. Dividiremos la construcción de este objeto en etapas, utilizando en cada momento la información provista por nuestro oponente. Nuestra estrategia debe ser diseñada de manera tal que resulte en la construcción del objeto deseado no importe lo que nuestro oponente haga.

Sea A n-c.e, entonces

Hecho 3.1.1. *Supongamos que σ_1 y σ_2 son dos cadenas distintas. Entonces no pueden haber etapas $s_1 < s_2 < \cdots < s_{n+2}$ tales que alternamos entre $\sigma_1 \preceq A_{s_i}$ y $\sigma_2 \preceq A_{s_{i+1}}$.*

Antes de comenzar con la demostación para el caso A n-c.e. repasaremos la demostración para el caso $n = 1$ intentando hallar de qué manera podemos extender este resultado.

Consideremos una función computable α tal que para todo e vale $J^\sigma(\alpha(e)) = \sigma$ para todo σ tal que $J^\sigma(e)\downarrow$. La existencia de tal función está dada por el hecho 2.2.1. Supongamos que queremos predecir la convergencia o divergencia de $J^A(e)$, como no disponemos de A sino de una aproximación comenzaremos prediciendo la divergencia hasta que $J^A(e)[s]\downarrow$ y el uso σ es trazado en $T_{\alpha(e)}$, donde $\{T_e\}_{e\in\mathbb{N}}$ constituye una traza c.e. de A. Hasta el momento las condiciones mencionadas anteriormente presentan un primer indicio de que podemos estar en lo correcto con lo cual nos inclinamos por la convergencia. Ahora bien, como disponemos de una aproximación de A puede que el uso σ no pertenezca a A, es decir, puede ser que en un paso posterior nuestra aproximación difiera de σ en por lo menos un bit, con lo cual nuestro cómputo $J^\sigma(\alpha(e)) = \sigma$ pierde validez. Si este es el caso entonces retomamos nuestra primera predicción: la divergencia. En el caso c.e. no perdemos nada puesto que tenemos la certeza de que la cadena σ no será extendida en el futuro, con lo cual forzamos a que nuestro oponente, en caso de que quiera hacernos creer en la convergencia

utilize otro lugar de la traza $T_{\alpha(e)}$ que sabemos es finita. De esta manera encontramos una estrategia exitosa. El caso Δ^0_2 presenta una dificultad insalvable para la estrategia anterior: nuestro oponente posee más libertad con lo cual nos puede hacer cambiar a convergencia con σ para luego cambiar nuestro conjunto para diferir en por lo menos un bit de σ lo cual nos forzaría a predecir divergencia y luego volver a σ y hacer esto cuantas veces lo desee, acabando de esta manera con nuestras esperanzas de que A' sea ω-c.e.

Sin embargo el fracaso de la estrategia anterior en el caso Δ^0_2 se debe básicamente al hecho de que no poseemos a priori un control uniforme de la cantidad de cambios de A. Veamos qué sucede si restringimos la capacidad del oponente a cambiar nuestro conjunto A a una cantidad fija de veces, digamos que A es 2-c.e. La estrategia anterior sigue sin funcionar puesto que nuestro oponente puede presentarnos un σ cambiarlo y luego puede volver a aparecer, sin embargo con ese σ en particular no puede hacer esto más que dos veces, en analogía al caso $n = 1$ donde se utilizaba una función computable α como un certificado, en el caso $n = 2$ pedimos dos certificados, es decir, si antes nos fiabamos de un cómputo demasiado rápido en el caso $n = 2$ seremos más cautelosos. La clave es requerir certificados no sólo a prefijos $\sigma \preceq A$ tales que $J^\sigma(e)\downarrow$ sino también a prefijos que todavía no han convergido. Nuestra estrategia será entonces la siguiente: de manera similar al caso $n = 1$ comenzaremos adivinando divergencia y cambiaremos a convergencia únicamente si en la etapa s $J^A(e)[s]\downarrow$ con uso σ y $\sigma \in T_{\alpha(e)}$. Ahora qué sucede si en una etapa posterior aparece τ incompatible con σ? En el caso $n = 1$ inmediatamente descartábamos σ como falso, en este caso verificamos antes de descartar a σ que $\tau \upharpoonright |n|$ pertenezca a una traza T_β con un β que será especificado en la demostración. Si esto sucede entonces cambiamos a divergencia. Si nuestro oponente retorna a la cadena σ entonces como A es 2-c.e. entonces podemos estar seguros de que τ no volverá aparecer con lo cual nuestro oponente ha gastado un elemento de la traza T_β. Notemos que si A fuese ω-c.e. nuestro oponente podría vencernos si utilizaramos la estrategia anterior puesto que podría elegir a su conveniencia la cantidad de veces a intercalar σ y τ, de hecho, Nies [25] demuestra el siguiente

Teorema 3.1.1. *Existe un conjunto ω-c.e y jump traceable que no es superlow.*

Luego de la discusión previa podemos demostrar de manera rigurosa el

Teorema 3.1.2. *Todo conjunto n-c.e. y jump traceable es superlow.*

Demostración. Supongamos que A es n-c.e y jump traceable. Sea $A = \cup_s A_s$ una aproximación computable de A tal que para todo x, $|\{s : A_{s+1}(x) \neq A_s(x)\}| \leq n$. Sea h una función de orden, y $\{T_x\}_{x\in\mathbb{N}}$ la traza c.e. para J^A, con cota h. Sea $\alpha(e)$ una función de reducción tal que para todo e, $J^\sigma(\alpha(e)) = \sigma$ para todo σ tal que $J^\sigma(e)\downarrow$. Nuestra intención es construir una aproximación ω-c.e. $g(e, s)$ de $A'(e)$ que esté acotada por una función $f(e)$ computable. Para ello necesitaremos funciones que certifiquen al menos parcialmente la estabilidad de los prefijos considerados, una de éstas funciones es $\alpha(e)$, la otra función es $\beta(e, k)$, definida como $J^\sigma(\beta(e, k)) = \sigma$ para todo σ tal que $|\sigma| = |t_{\alpha(e),k}|$, donde $t_{e,k}$ es el k-ésimo elemento enumerado en T_e en caso de existir.

Comenzaremos definiendo $g(e,0)=0$ y nuestra aproximación permanecerá estable, es decir $g(e,s)=g(e,s+1)$, a menos que en la etapa s ocurra un evento que nos haga cambiar de parecer. Los eventos que pueden alterar la aproximación son:

1. Cambiamos de $0 \to 1$ en la etapa s si $J^A(e)[s]\downarrow$ con uso σ y $\sigma \in T_{\alpha(e)}$.

2. Cambiamos de $1 \to 0$ en la etapa s si $A_s \upharpoonright |\sigma| \in T_{\beta(e,k)}$ y además $A_s \upharpoonright |\sigma| \neq \sigma$ donde $\sigma = t_{\alpha(e),k}$ y σ produjo el cambio anterior de $0 \to 1$.

Verificación: Sea e fijo. Contaremos el número de cambios. Sean $s_1 < s_2 < \cdots < s_N$ todas las etapas en que cambiamos de 1 a 0. Notemos que para cada $m \leq N$ debemos tener $A_s \upharpoonright |\sigma| \in T_{\beta(e,k)}$ donde $\sigma = t_{\alpha(e),k}$ y σ produjo el cambio anterior de $0 \to 1$. Notemos a su vez que $A_{s_m} \upharpoonright |\sigma| \neq \sigma$. Asociaremos con la etapa s_m el par $\langle \sigma, A_{s_m} \upharpoonright |\sigma| \rangle$. Más precisamente definimos $P(m) = \langle \sigma, A_{s_m} \upharpoonright |\sigma| \rangle$.

Supongamos que para cierto $m' > m$ tenemos $P(m') = P(m) = \langle \sigma, \tau \rangle$. Luego, claramente en una etapa t con $s_m < t < s_{m'}$ cambiamos de 0 a 1 y $\sigma \subset A_t$. Como $\sigma \neq \tau$ y A es n-c.e. se sigue que la cardinalidad de $\{m \leq N : P(m) = \langle \sigma, \tau \rangle\}$ para cada $\langle \sigma, \tau \rangle$ es como mucho n. Nos preguntamos cuántos pares de estos pueden haber? Como tanto σ como τ deben pertencer a la traza la cantidad de pares está gobernada por h con lo cual el número total de pares distintos es como mucho $h(\alpha(e)) \cdot h(\beta(e, h(\alpha(e)))$ con lo cual el número de cambios de $g(e,\cdot)$ es $\leq 2N$ y estará acotado por $f(e) = 2n \cdot h(\alpha(e)) \cdot h(\beta(e, h(\alpha(e))))$

Verificaremos ahora que $\lim_s g(e,s) = A'(e)$. Si $J^A(e)\downarrow$ con uso σ entonces el límite $\lim_s g(e,s)$ no puede ser 0 puesto que $J^A(\alpha(e)) = \sigma$ y en consecuencia σ debe aparecer en $T_{\alpha(e)}$. Recíprocamente si $\lim_s g(e,s) = 1$ luego en una etapa final s cambiamos $g(e,\cdot)$ de 0 a 1, luego de observar que $J^A(e)[s]\downarrow$ con uso σ, donde $\sigma = t_{\alpha(e),k}$. Afirmamos que $\sigma \subset A$. Notemos que no requerimos que $\sigma \subset A_t$ para todo $t > s$ a menos que $n = 1$. Supongamos luego que $\sigma \not\subset A$, entonces se debe haber dado $J^A(\beta(e,k)\downarrow) = A \upharpoonright |\sigma| \neq \sigma$ y éste prefijo eventualmente aparecerá en $T_{\beta(e,k)}$ lo cual producirá un cambio en $g(e,\cdot)$ luego de haberse estabilizado, lo cual resulta una contradicción. □

3.1.2. Todo conjunto n-c.e y superlow es jump traceable

Supongamos que A es n-c.e. y superlow. Sea $A = \cup_s A_s$ una aproximación de A tal que para todo x, $|s : A_{s+1}(x) \neq A_s(x)| \leq n$. Como A es superlow entonces tenemos funciones computables g y h tales que para todo e, $\lim_s g(e,s) = A'(x)$ y $|s : g(e,s) \neq g(e,s+1)| \leq h(e)$. El número de cambios de $g(e,s)$ hasta la etapa s se denotará $ch(e,s)$ y será más precisamente el x más grande tal que $2(x-1) \leq |t < s : g(e,s) \neq g(e,s+1)|$, por otro lado se denotará como $ch(e)$ al número total de cambios.

Descripción de la estrategia Nuevamente describiremos primero el caso $n = 1$. Nuestra intención es enumerar posibles cómputos del salto $J^A(e)$ en la traza X_e dado que nuestro oponente nos provee de una aproximación $g(x,s)$ de A' (A es superlow). Enumeraremos también el funcional con índice v, donde v es dado por el Teorema de la Recursión 2.2.3.

Cada vez que vemos que $J^{\tau}(e)[s]\downarrow$ con $\tau \preceq A_s$ copiaremos el cómputo estableciendo que $J^{\tau}(v)\downarrow$. Supongamos que luego de que nuestro oponente nos muestra los cómputos con usos $\tau_1, \tau_2, \ldots, \tau_n$ decide cambiar $g(v,\cdot)$ de 0 a 1, entonces creeremos en el valor del salto actual y pondremos $J^{\tau_n}(e)[s]$ en X_e. Si A es c.e. nuestro oponente no puede volver a presentarnos ningún τ_i para $i < n$, razón por la cual trazaremos únicamente un valor, $J^{\tau_n}(e)[s]$, de éste lote de cómputos. Si posteriormente el oponente decide mostrarnos otro lote de cómputos, sólo creeremos en esta nueva evidencia en caso de que nuestro oponente se vea obligado a cambiar g de $1 \to 0 \to 1$ nuevamente. Con lo cual resulta claro que $|X_e| \leq h(v)$, donde h es una cota para g.

Resulta fundamental el hecho de que A sea c.e. puesto que una vez que nuestro oponente ha abandonado un prefijo no puede volver sobre él. El problema en el caso Δ^0_2 es el siguiente: Se nos puede presentar cómputos con usos $\tau_1, \tau_2, \ldots, \tau_n$ en un primer lote, para cambiar luego g de 0 a 1, manteniéndose en 1 podría mostrarnos los cómputos en el orden inverso, $\tau_n, \ldots, \tau_2, \tau_1$, en cada momento nuestro oponente espera a que tracemos $J^{\tau_i}(e)$ y luego se mueve al próximo $\tau_{i-1} \subset A_s$. En este caso nos veríamos forzados a ingresar todos los valores de los cómputos en X_e y toda la pérdida del oponente sería un único cambio en $g(e,\cdot)$. Como es nuestro oponente quien decide qué tan grande es el lote de cómputos, la estrategia anterior se encuentra condenada al fracaso. Qué ideas podemos adoptar del caso $n = 1$ para al caso n-c.e? Notemos que en el caso más sencillo cada prefijo pasa por dos tipos de certificaciones: en un primer paso nuestro algoritmo se encarga de seleccionar candidatos pidiendo que el prefijo τ converja en la etapa s, es decir $J^{\tau}(e)[s]$ con $\tau \subset A_s$. Esta primera certificación hace plausible que sea un valor final, sin embargo no nos decidimos sino hasta que obtenemos nuestra segunda certificación, es decir g cambia de 0 a 1, lo cual acota con h la libertad de elección. En el caso $n = 2$ la estrategia anterior falla puesto que prefijos pasados pueden volver. En consecuencia nuestros dos certificados resultan escasos y es necesario incluir más *tests*. La pregunta es, por cuántas certificaciones tendrá que pasar un prefijo hasta que podamos creer en él? Veamos el caso $n = 2$. Supongamos que se nos presente el siguiente lote de cómputos $\tau_1, \tau_2, \ldots, \tau_N$ y luego se nos muestra el lote anterior en el orden inverso, es decir $\tau_N, \tau_{N-1}, \ldots, \tau_1$ una vez que ha hecho esto no puede volver a mostrarnos $\tau_i \subset A_s$ para $i > 1$ puesto que A es 2-c.e. Este es el hecho que nos dará un indicio de cuántas veces debemos testear un prefijo τ antes de creer en él.

En el caso $n = 1$ nos servíamos del índice v de un funcional, en el caso $n = 2$ tendremos mediante el Teorema de la Recursión un segundo índice v_1. El propósito de este índice es testear A cuando el oponente regresa a un prefijo utilizado con anterioridad. Esto es, aguardamos el primer lote de cómputos $\tau_1, \tau_2, \ldots, \tau_N$ y luego el cambio de $g(v,\cdot)$ de 0 a 1 Si el prefijo τ_N persiste a medida que las etapas avanzan entonces definimos $J^{\tau_N}(v_1)\downarrow$ y esperamos un cambio de $g(v_1,\cdot)$, es decir, testeamos dos veces a nuestro candidato. En esta situación nuestro oponente puede, o bien cambiar $g(v_1,\cdot)$ de 0 a 1, en cuyo caso ingresamos $J^{\tau_N}(e)$ a X_e o abandonar τ_N, si decide presentarnos τ_{N-1} entonces hemos avanzado en nuestra estrategia puesto que no hemos ingresado nada innecesario en X_e y sabemos que no puede volver a presentarnos τ_N, en ese caso definimos $J^{\tau_{N-1}}(v_1)\downarrow$ y repetimos el procedimiento. Cuando finalmente alcanza τ_1 el número total de valores que podemos haber ingresado en X_e está acotado por $h(v_1)$ pero este lote se encuentra completamente

exhausto puesto que no puede volver a aparecer ningún prefijo τ_i para $i \leq N$.

Notemos que en cualquier momento nuestro oponente puede abandonar el primer lote de cómputos que nos presentó y mostrarnos uno nuevo $\tau'_1, \ldots, \tau'_N$ que resute incompatible con el lote anterior. En este caso, esperaremos que cambie $g(v, \cdot\,)$ de $1 \to 0 \to 1$ para luego repetir el mismo procedimiento empleado en el primer paso. La cota total sobre X_e será $h(v_1) + \cdots + h(v_{h(v)})$.

Una idea similar puede utilizarse en el caso general. Lo fundamental es conocer a priori la cantidad de *tests* que debe pasar un prefijo particular y ser cuidadoso a la hora de organizar como certificamos los distintos prefijos, únicamente cuando un prefijo ha pasado todas las pruebas podemos con seguridad ingresarlo a X_e de manera de poder controlar la cota. Con el objeto de organizar nuestra estrategia utilizaremos un árbol de índices.

El árbol de índices. De manera similar al caso $n = 1$ enumeraremos funcionales de Turing cuyos índices nos resultan conocidos anticipadamente gracias al Teorema de la Recursión. Con el objeto de trazar de manera adecuada $J^A(e)$ necesitaremos n capas de índices. Representaremos nuestra estrategia como un árbol finito T^e de secuencias

$$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$$

con $m \leq n - 1$ y $\sigma_i \in \mathbb{N}$. Cada nodo $\sigma \in T^e$ tiene asociado un índice v_σ y el nodo λ tiene asociado el índice v_λ. Supongamos que σ es tal que $|\sigma| < n - 1$ y su índice correspondiente es v_σ, entonces σ tendrá $h(v_\sigma)$ sucesores cuyos índices asociados serán $v_{\sigma^\frown 1}, v_{\sigma^\frown 2}, \ldots, v_{\sigma^\frown h(v_\sigma)}$ para cada nodo. Construiremos una traza c.e. X_e que estará acotada por $|X_e| \leq H(e) = \sum_{\sigma \in T^e} h(v_\sigma)$. Enumeraremos axiomas $J^\sigma(v_\sigma)$ para todo $\sigma \in T^e$ que nos ayudarán a determinar que ciertos prefijos son los adecuados. Controlar los valores de $J^\sigma(v_\sigma)$ nos resulta innecesario puesto que $g(v_\sigma, \cdot\,)$ sólo adivina convergencia o divergencia, por esta razón estableceremos que los valores de los funcionales sean un conjunto V_σ libre de prefijos que representan los usos. Esto es, la acción de definir el axioma $J^\tau(v_\sigma)\downarrow$ equivale a agregar el elemento τ a V_σ.Definiremos el conjunto V_σ que determina nuestro funcional, como la unión disjunta de subconjuntos

$$V_\sigma = V_{\sigma,1} \cup \ldots \cup V_{\sigma,h(v_\sigma)}$$

Donde ingresaremos elementos en $V_{\sigma,k+1}$ si nuestra función g ha cambiado de parecer k veces.

Construcción de X_e. Cada X_e será construido de manera independiente pero de manera uniforme. El hecho de que describamos nuestro funcional como una unión de conjuntos nos impone ciertas condiciones de consistencia que pueden describirse de la siguiente manera

$$(\dagger) \text{ si } \sigma_1 \frown k \subset \sigma_2, \text{ entonces } V_{\sigma_2} \subseteq V_{\sigma_1,k}$$

En la etapa $s = 0$ para todo σ y k, $V_{\sigma,k} = \emptyset$. Supongamos que $s > 0$. Utilizaremos como referencia el árbol T^e para guiar nuestros pasos en la etapa s. En todo momento nuestro algoritmo hará referencia a un nodo en particular $\sigma \in T^e$ que llamaremos $\diamond$ y que irá cambiando de acuerdo a ciertas condiciones. En el comienzo de cada etapa s nos posicionaremos en la ráiz del árbol, es decir $\diamond := \lambda$ y avanzaremos por el árbol de acuerdo al siguiente esquema:

1. $g(v_\diamond, s) = 0$: nos fijamos si $J^A(e)[s]\downarrow$. En caso de que esto suceda ingresamos el uso τ en $V_{\sigma,ch(v_\diamond,s)}$ a menos que τ ya se encuentre en $V_\diamond$. En cualquier caso detengo el procedimiento y continúo a la próxima etapa.

2. $g(v_\diamond, s) = 1$ y A_s no extiende a ningún $\tau \in V_\sigma$: en este caso nuestra única acción consiste en avanzar al paso siguiente.

3. $g(v_\diamond, s) = 1$ y $\tau \subset A_s$ para algún $\tau \in V_{\diamond,k}$: Si $|\diamond| = n-1$ nos encontramos en una hoja del árbol T^e, es decir nuestro prefijo posee todos los certificados necesarios y podemos creer en $J^\tau(e)$, con lo cual lo enumeramos en X_e. En este caso detenemos el paso y avanzamos a la próxima etapa. Si $|\diamond| < n-1$ reasignamos $\diamond$ a $\diamond \frown k$, es decir, avanzamos al próximo nivel. Luego, repetimos el procedimiento desde el principio con el nuevo valor de $\diamond$.

Verificación: como cada V_σ copia los usos de cómputos convergentes $J^A(e)[s]$ resulta claro que es un conjunto libre de prefijos. Claramente los conjuntos $V_{\sigma,1}, \cdots, V_{\sigma(h(v_\sigma))}$ son disjuntos dos a dos y se cumple (†). Como $ch(v_\sigma) \leq h(v_\sigma)$ vale en todo momento, se sigue que no hemos hecho ninguna asignación ilegal durante la construcción. El siguiente lema resulta fundamental en cuanto utiliza la hipótesis de que A es un conjunto n-c.e. Básicamente el lema afirma que si τ_1 ingresa en $V_{\sigma,k}$ para un σ tal que $|\sigma| = n-1$ antes que un τ_2, entonces, si τ_2 ingresa en $V_{\sigma,k}$ no podemos volver a tener que $\tau_1 \subset A_s$.

Lema 3.1.3. *Fijemos un número k y una secuencia σ tal que $|\sigma| = n-1$. Supongamos que $\tau_1 \neq \tau_2$ son ingresados en $V_{\sigma,k}$ en las etapas $u < u'$ respectivamente. Luego para todo $t > u'$ no podemos tener $\tau_1 \subset A_t$.*

Demostración. Sea $\sigma = (k_1, k_2, \ldots, k_{n-1})$ y $\sigma_i = \sigma \upharpoonright i$. Notemos que por (†) tanto τ_1 como τ_2 deben pertenecer a $V_{\sigma_i,k_{i+1}}$ para todo $i < n-2$, es decir, para todos los predecesores de σ. Con lo cual se sigue que existen etapas $s_0 < s_1 < \cdots < s_{n-2} < u$ y $t_0 < t_1 < \cdots < t_{n-2} < u'$ tales que τ_1 ingresa a $V_{\sigma_i,k_{k+i}}$ en la etapa s_i y τ_2 ingresa a $V_{\sigma_i,k_{k+i}}$ en la etapa t_i. Nuestra intención es intercalar las etapas s_i y t_i de manera tal de obtener una sucesión $\tau_1, \tau_2, \ldots \tau_1, \tau_2$ suficientemente larga como para poder usar el hecho de que A es n-c.e. Para poder crear una tal sucesión debemos primero probar que esto es posible, es decir veremos que vale $\max\{s_i, t_i\} < \min\{s_{i+1}, t_{i+1}\}$. Fijemos un $i < n-2$ y sea $s' = \min\{s_{i+1}, t_{i+1}\}$. Notemos que no se hace ninguna otra enumeración en $V_{\sigma_i,k_{i+1}}$ en la etapa s' o luego de ella debido a que en s' tenemos que $V_{\sigma_i,k_{i+1}}[s'] \neq \emptyset$ así como también que $g(v_{\sigma_i}, s') = 1$ y $ch(v_{\sigma_i}, s') \geq k_{i+1}$. Cualquier otra enumeración que se haga en V_{σ_i} debe esperar hasta que $g(v_{\sigma_i}, \cdot\,)$ cambie a 0 nuevamente, con lo cual $ch(v_{\sigma_i})$ pasará a ser $> k_{i+1}$. En consecuencia, tanto τ_1 como τ_2 deben encontrarse en $V_{\sigma_i,k_{i+1}}$ en la etapa s'.

Elijamos ahora la secuencia adecuada de etapas s de manera tal que vamos alternando entre $\tau_1 \subset A_s$ y $\tau_2 \subset A_s$. Existen por lo tanto n cambios y por el hecho 3.1.1 no pueden haber más. □

Queremos demostrar que $|X_e| \leq h'(e)$. Para ello bastará demostrar que para cada $\sigma \in T^e$ tal que $|\sigma| = n-1$ y para cada k tenemos que $|\{\tau : \tau \in V_{\sigma,k} \wedge J^\tau \in X_e\}| \leq 1$. Este

hecho se sigue de manera inmediata del lema 3.1.3. Finalmente sólo nos resta demostrar que si $J^A(e) \downarrow = q$ con uso τ entonces $q \in X_e$. Definamos el nodo σ de largo $|\sigma| = n-1$ (es decir una hoja) de manera inductiva, de manera tal que para cada $i \leq n-1$, $\sigma_i = \sigma \upharpoonright i$ y además valen las siguientes propiedades:

1. $\lim_s g(v_{\sigma_i}, s) = 1$,
2. $\tau \in V_{\sigma_i, \sigma(i)}$,
3. para casi toda etapa t, $\diamond$ visitará el nodo σ_i en la etapa t.

Asumamos luego que σ_i posee las propiedades mencionadas con anterioridad y $\sigma_{i+1} = \sigma_i \frown \sigma(i)$ está definido. Continuamos nuestra inducción demostrando que σ_{i+1} también posee las propiedades deseadas y define entonces $\sigma(i+1)$. Resulta claro que en la etapa posterior a que $A \upharpoonright |\tau|$ se estabiliza (digamos en u), $\diamond$ visitará el nodo σ_{i+1}. En este caso no podemos tener $g(v_{\sigma_{i+1}}, s) = 0$ puesto que si así fuese esto implicaría que g predice A' de manera incorrecta. En consecuencia $\lim_s g(v_{\sigma_{i+1}}, s) = 1$ y se sigue que eventualmente τ debe ser enumerado en V_{σ_i+1} puesto que si no sucediese esto tendriámos que $J^A(v_{\sigma_i+1}) \uparrow$ lo cual implicaría que $\lim_s g(v_{\sigma_{i+1}}, s) = 1$ es incorrecto. Luego $\tau \in V_{\sigma_{i+1},k}$ para cierto k y elegimos $\sigma(i+1) = k$. Esto completa la inducción. Finalmente, como un σ existe con las propiedades requeridas se sigue que eventualmente $J^\tau(e) = q$ será enumerado en X_e como se deseba.

Capítulo 4

Strong Jump Traceability

Recordemos que un conjunto A c.e. se llama promptly simple si A es co-infinito y existe una función recursiva p y una aproximación $(A_s)_{s\in\mathbb{N}}$ de A tal que, para cada e,

$$|W_e| = \infty \Rightarrow \exists s \exists e[x \in W_{e,s+1} \setminus W_{e,s} \wedge x \in A_{p(s)}]$$

Notemos que se puede demostrar que los conjuntos c.e. promptly simple son no computables. En esta sección se introducirá una noción de jump-traceability más fuerte y se verá que existe un conjunto promptly simple que es strongly jump-traceable.

4.1. Existencia y propiedades básicas

Definición 4.1.1. Se dirá que un conjunto A es strongly jump-traceable sii para cada función de orden h, A es jump-traceable vía h.

Proposición 4.1.2. *$\{A$: A es strongly jump-traceable$\}$ es cerrado hacia abajo respecto a reducibilidad de Turing.*

Demostración. Supongamos que A es strongly jump-traceable y $B \leq_T A$. Demostraremos que B es jump-traceable vía la función de orden h. Sea Ψ el funcional tal que $\Psi^A(x) = J^B(x)$ para todo x y sea α su función de reducción, de manera que $J^A(\alpha(x)) = \Psi^A(x)$. Sabemos que A es jump-traceable vía la traza $(T_i)_{i\in\mathbb{N}}$ con cota g, donde $g(z) = h(\text{mín}\{y : y \in \mathbb{N} \wedge \alpha(y+1) \geqslant z\})$. Observemos que como α es una función de orden entonces g también lo es. Luego, resulta claro que

$$J^B(e) = J^A(\alpha(e)) \downarrow \Rightarrow J^B(e) \in T_{\alpha(e)}.$$

Ahora, $g(\alpha(e)) = h(y)$ para cierto y tal que $\alpha(y) < \alpha(e)$ ó $y = 0$. Luego $y \leq e$ y $g(\alpha(e)) = h(y) \leq h(e)$. En consecuencia $(T_{\alpha(i)})_{i\in\mathbb{N}}$ es una traza para el salto de B con cota h.

Resulta claro que un conjunto computable A resulta strongly jump-traceable puesto que podemos dar la siguiente traza

$$T_e = \begin{cases} \{J^A(e)\} & \text{si } J^A(e)\downarrow; \\ \emptyset & \text{caso contrario} \end{cases}$$

□

En el próximo teorema 4.1.4 se demostrará la existencia de un conjunto no computable que es strongly jump-traceable. Para ello necesitamos el próximo lema, demostrado en [17, Teorema 2.3.1]

Lema 4.1.3. *La función* $m(x) = \min\{C(y) : y \geqslant x\}$ *es no acotada, no decreciente y para cada función de orden* f *existe un* x_0 *tal que* $m(x) < f(x)$ *para todo* $x \geqslant x_0$. *Además* $m(x) = \lim_{s\to\infty} m_s(x)$, *donde* $m_s(x) = \min\{C_s(y) : x \leq y \leq x + s\}$ *es recursiva y* $m_s(x) \geqslant m_{s+1}(x)$, *para todo* x *y* s.

Teorema 4.1.4. *Existe un conjunto strongly jump-traceable y promptly simple.*

Demostración. Construiremos un conjunto A promptly simple en etapas de manera que satisfaga los siguientes requerimientos

$$P_e : |W_e| = \infty \Rightarrow \exists s \exists x[x \in W_{e,s+1} \setminus W_{e,s} \wedge x \in A_{s+1}]$$

Con estos requerimientos nos aseguraremos de que A sea promptly simple. Cada vez que enumeremos un elemento en A con el objeto de cumplir un requerimiento puede que destruyamos con esto uno de los cómputos $J^A(k)$, es decir la introducción de un nuevo elemento en A puede que altere el valor de $J^A(k)$, esto último hará que nuestra traza crezca. Será necesario por lo tanto, controlar qué elementos entran a la traza y cuáles quedan afuera. Como por la definición de strongly jump-traceable necesitamos conseguir una traza para J^A acotada por cualquier función de orden h trabajaremos con la función m definida en el lema 4.1.3 la cual crece más lento que cualquier otra función de orden. La construcción se hará de manera tal que P_e destruirá $J^A(k)$ en la etapa s sólo si $e < m_s(k)$ (recordemos que como $m_s(k) \geqslant m_{s+1}(k)$ esta restricción impuesta en P_e puede ir fortaleciéndose a medida que s crece). De esta manera nos aseguramos de que la traza $J^A(e)$ permanece acotada por $m(e)$ lo cual resultará suficiente pues $m \leq h$ a partir de un momento. Se verá posteriormente que la elección de la traza para J^A no se realiza de una manera uniforme sino que depende de h.

En lo que sigue, asumiremos que $W_{e,s} \subseteq \{0, 1, \ldots, s\}$ para todo índice e y etapa s. *Construcción de A.* Sea m_s la función no acotada y no decreciente definida en el lema 4.1.3

Etapa 0**:** Definimos $A_0 = \emptyset$ y declaramos P_e no satisfecho para ningún e.

Etapa $s + 1$**:** elijamos el mínimo $e \leq s$ tal que

- P_e aún no se ha satisfecho;
- Existe un x tal que $x \in W_{e,s+1} \setminus W_{e,s}, x > 2e$ y para todo k tal que $m_s(k) \leq e$, si $J^A(k)[s]$ está definido, entonces x es mayor que el uso de $J^A(k)[s]$

Si un tal e existe, ponemos en A el menor x correspondiente a cada e. Decimos que A recibe atención en la etapa $s+1$ y declaramos el requerimiento P_e satisfecho. En otro caso, $A_{s+1} = A_s$. Finalmente definimos, $A = \cup_s A_s$

Verificación. Resulta claro que P_e recibe atención como mucho una vez, este hecho nos permite deducir que cada requerimiento influye en la construcción de A en una única etapa.

Para demostrar que A es strongly jump-traceable, fijamos una función de orden h. Se verá que existe una traza c.e. T para J^A. Sea h una función de orden arbitraria, por el lema 4.1.3 existe k_0 tal que para todo $k \geqslant k_0, m(k) < h(k)$. Definamos la función recursiva

$$f(x) = \begin{cases} \text{mín}\{s : m_s(k) \leq h(k)\} & \text{si } k \geqslant k_0 \\ 0 & \text{caso contrario.} \end{cases}$$

Para $k \geqslant k_0$ y $s \geqslant f(k), m_s(k)$ estará por debajo de $h(k)$, con lo cual $J^A(k)$ puede cambiar porque P_e recibe atención, para $e < m_s(k) \leq h(k)$. Como cada P_e recibe atención como mucho una vez, $J^A(k)$ puede cambiar como mucho $h(k)$ veces luego de la etapa $f(k)$. Con lo cual

$$T_k = \begin{cases} \{J^A(k)[s] : J^A(k)[s]\downarrow \wedge s \geqslant f(k)\} & \text{si } k \geqslant k_0; \\ \{J^A(k)\} & \text{si } J^A(k)\downarrow \wedge k < k_0 \\ \emptyset & \text{en otro caso} \end{cases}$$

es la traza deseada.

Fijemos un e tal que W_e es infinito y veamos que P_e se satisface. Sea s tal que

$$\forall k[m(k) \leq e \Rightarrow m_s(k) = m(k)]$$

y $s' > s$ tal que ningún P_i recibe atención luego de la etapa s' para cualquier $i < e$. Luego, por construcción, ningún cómputo $J^A(k), m(k) \leq e$ puede ser destruído luego de la etapa s'. Con lo cual existe $t > s'$ tal que para todo k donde $m_t(k) \leq e$, si $J^A(k)$ convenverge, entonces el cómputo es estable de la etapa s en adelante. Elijamos $t' > t$ tal que existe un $x \in W_{e,t'+1} \setminus W_e, t', x > 2e$ y x es mayor al uso de todos los $J^A(k)$ convergentes para todo k donde $m_{t'}(e) \leq e$. Ahora bien P_e ya fue tenido en cuenta o P_e recibe atención en la etapa $t' + 1$. En ambos casos el requerimiento P_e es satisfecho. □

4.2. Aproximaciones del salto

Definición 4.2.1. Un conjunto D se dirá que es *well-aproximable* sii para cada función de orden b, D es ω-c.e. vía b. Diremos también que un conjunto A es *strongly superlow* si A' es well-aproximable.

Proposición 4.2.2. *$\{A{:}A'$ es well-aproximable$\}$ es cerrado hacia bajo respecto a reducibilidad de Turing.*

Demostración. Supongamos que A es tal que A' es well-aproximable y $B \leq_T A$.Veremos que B' es well-aproximable vía la función de orden dada b. Definamos Ψ y α como en la proposición 4.1.2. Sabemos que existe una función recursiva g a valores en $\{0,1\}$ tal que

$A'(x) = lim_s g(x,s)$ y $g(x,s)$ cambia al menos $h(x)$ veces, donde $h(x) = b(min\{y : y \in \mathbb{N} \wedge \alpha(y+1) \geqslant z\})$. Observemos que como b es una función de orden h también lo es. Se sigue entonces que

$$\lim_{s\to\infty} g(\alpha(x), s) = A'(\alpha(x)) = B'(x)$$

y $g(\alpha(x), s)$ cambia como mucho $h(\alpha(x))$ veces. Como en la Proposición 4.1.2 se ve que $h(\alpha(x)) \leq b(x)$. □

En lo que sigue probaremos que si A es c.e. entonces A es strongly jump-traceable sii A es strongly superlow. Para ello necesitaremos los siguientes lemas.

Lema 4.2.3. *Sean f y g dos funciones de orden tales que $f(x) \leq g(x)$ para casi todo x*

(a) Si A es jump-traceable vía f entonces A es jump-traceable vía g

(b) Si A es well-aproximable vía f entonces A es well-aproximable vía g

Demostración. Asumamos que $\exists x_0 \forall x[x \geqslant x_0 \Rightarrow f(x) \geqslant g(x)]$. Para (a), supongamos que T es una traza para J^A con cota f. Podemos definir la traza T':

$$T'_x = \begin{cases} T_x & \text{si } x \geqslant x_0 \\ \{J^A(x)\} & \text{caso contrario} \end{cases}$$

Luego, si $x \geqslant x_0$ entonces $|T'_x| = |T_x| \leq f'(x)$, y si $x < x_0$ entonces $1 = |T'_x| \leq f'(x)$. Para (b), supongamos que A es well-approximable vía la función $g(x,s)$ a valores en $\{0,1\}$ que cambia como mucho $f(x)$ veces. Definimos

$$g'(x) = \begin{cases} g(x,s) & \text{si } x \geqslant x_0, \\ A(x) & \text{caso contrario .} \end{cases}$$

Si $x \geqslant x_0$ luego $g'(x,s)$ cambia como mucho $f(x) \leq f'(x)$ veces, y si $x > x_0$ entonces g' no cambia. □

Lema 4.2.4. *Existe una función recursiva γ tal que para todo conjunto c.e. A vale:*

(a) Si A es jump-traceable vía una función de orden h entonces A es super-low vía una función de orden $b(x) = 2h(\gamma(x)) + 2$;

(b) Si A es super-low vía una función de orden b entonces A es jump-traceable vía una función de orden $h(x) = b(\lfloor \frac{1}{2}\gamma(x) \rfloor)$

Demostración. $(a) \Rightarrow (b)$ Supongamos que A es jump-traceable vía h. Por [25] A es superlow vía una función g a valores en {0,1} tal que $g(x,s)$ cambia como mucho $2h(\alpha(x)+2$ veces, donde α es la función de reducción (y por lo tanto primitiva recursiva) que depende de A. La diagonal de la función de Ackermanann γ satisface $\gamma(x) \geqslant \alpha(x)$ para casi todo

x [27, Volúmen II, Teorema VIII.8.10]. Como h es una función de orden, $2(h \circ \gamma) + 2$ también lo es, y $2h(\gamma(x)) + 2 \geqslant h(\alpha(x)) + 2$ para casi todo x. Por el lema 4.2.3 A resulta super-low vía $b(x) = 2h(\gamma(x)) + 2$. $(b) \Rightarrow (a)$ Supongamos que A es super-low vía una función de orden b y la función g. Siguiendo nuevamente [25], existe una traza para J^A vía $\lfloor \frac{1}{2}(b \circ \gamma) \rfloor$, para una función primitiva recursiva α que depende de g. De manera similar a la implicación previa vemos que $\lfloor \frac{1}{2} b(\gamma(x)) \rfloor \geq \lfloor \frac{1}{2} b(\alpha(x)) \rfloor$ para casi todo x. En consecuencia A es jump-traceable vía $h(x) = \lfloor \frac{1}{2} b(\gamma(x)) \rfloor$. □

Teorema 4.2.5. *Sea A un conjunto c.e. Luego resultan equivalentes*

(a) A es strongly jump-traceable

(b) A es strongly super-low

Demostración. $(a) \Rightarrow (b)$. Dada la función de orden b, veamos que A es super-low vía h. Por la parte (a) del lema 4.2.4, resulta suficiente definir una función de orden h tal que $2h(\gamma(x)) + 2 \leq b(x)$ para casi todo x. Si $b(x) \geqslant 4$ entonces definimos $h(\gamma(x) = \lfloor \frac{b(x)-2}{2} \rfloor$ y si $b(x) < 4$, definimos $h(\gamma(x)) = 1$. Como γ se puede tomar estrictamente monótona, la definición anterior es correcta y la podemos completar para hacer de h una función de orden.

$(b) \Rightarrow (a)$. Dada una función de orden h veremos que A es jump-traceable vía h. Por la parte (b) del lema 4.2.4 bastará definir una función de orden b tal que $\lfloor \frac{1}{2} b(\gamma(x)) \rfloor \leq h(x)$ para casi todo x. El argumento final es similar al caso anterior. □

4.3. Trazabilidad y complejidad simple de Kolmogorov

Se dará una caracterización de los conjuntos strongly-jump-traceables en términos de la complejidad plana de Kolmogorov. Demostraremos como primer paso que si A' es well-aproximable entonces A satisface una condición que involucra a la complejidad de Kolmogorov y de ésta última relación se deducirá que para cualquier conjunto A tal que A' es well-aproximable, A resulta strongly-jump-traceable.

Teorema 4.3.1. *Si A' es well-approximable luego para cada función de orden h y para casi toda x vale: $C(x) \leq C^A(x) + h(C^A(x))$*

Demostración. La idea de la demostración es la siguiente: Sea h una función de orden arbitraria. Supongamos luego que q_x es un $A-programa$ minimal para x, es decir $C^A(x) = |q_x|$ y $C^A(q_x) = x$. Sabemos que existe una constante c tal que $C(x) \leq |q_x| + 2C(x|q_x) + c$. Como $|q_x| = C^A(x)$, sólo necesitamos demostrar que $2C(x|q_x) + c \leq h(|q_x|)$ para casi todo x. Dado q_x y el valor de $C(x|q_x)$, resulta posible hallar un programa p_x de largo $C(x|q_x)$ que describe a x con la ayuda de q_x, esto es $\tilde{U}(p_x, q_x) = x$. Puede demostrarse la existencia de una función recursiva a valores en $\{0, 1\}$ que aproxime los bits de p_x y cambie pocas veces (en la demostración esto se logrará mediante la introducción del funcional

Ψ). Luego, existe una descripción de x mediante los valores de $C(x|q_x)$, q_x y p_x. Podemos representar a p_x mediante una aproximación a $\{0,1\}$-valores. Esto nos llevará a concluir que $C(x|q_x) \leq 2|h(|q_x|)| + O(1)$, desigualdad que resulta suficiente para obtener la cota sobre $2C(x|q_x) + c$.

Sea $\Psi^A(m,n,q)$ un funcional que hace lo siguiente:

(a) Computar $x = U^A(q)$. Si $U^A(q) \uparrow$ luego $\Psi^A(m,n,q) \uparrow$;

(b) Hallar el primer programa p tal que $|p| = n$ y $\tilde{U}(p,q) = x$. Si no existe tal p luego $\Psi^A(m,n,q) \uparrow$;

(c) En caso de que $m \notin [1,n]$ luego $\Psi^A(m,n,q) \uparrow$. En otro caso, si el m-ésimo bit de p es 1 luego $\Psi^A(m,n,q) \downarrow$; y si esto no sucede definir $\Psi^A(m,n,q) \uparrow$.

Sea α una función de reducción tal que $J^A(\alpha(m,n,q)) = \Psi^A(m,n,p)$. Elijamos luego una función de orden b tal que $b(\alpha(n,n,q)) \leq nh(|q|)$ para todo n,q. Podemos aproximar $A'(x)$ con una función recursiva a valores en $\{0,1\}$ cuyos cambios estén acotados por $b(x)$.

Sea q_x un programa $A - minimal$ para x, esto es $U^A(q_x) = x$ y $|q_x| = C^A(x)$. Sea $n_x = C(x|q_x)$. Luego $\Psi^A(m,n_x,q_x) \downarrow$ sii el m-ésimo bit de p_x es 1, donde p_x es el primer programa tal que $|p_x| = n_x$ y $\tilde{U}(p_x,q_x) = x$.

Como A' es $\omega - c.e.$ vía b, luego:

$$p_x = A'(\alpha(1,n_x,q_x)) \ldots A'(\alpha(n_x,n_x,q_x))$$

cambia como mucho

$$\begin{aligned} n_x \,\text{máx}\{b(\alpha(m,n_x,q_x)) : 1 \leq m \leq n_x\} &\leq n_x b(\alpha(n_x,n_x,q_x)) \\ &\leq n_x^2 h(|q_x|) \end{aligned}$$

Como $\tilde{U}(p_x,q_x) = x$ y podemos describir p_x con n_x, q_x y el número de cambios de $A'(\alpha(1,n_x,q_x) \ldots A'(\alpha(n_x,n_x,q_x))$, tenemos que

$$\begin{aligned} n_x = C(x|q_x) &\leq 2|n_x| + |n_x^2 h(q(|x|)| + O(1) &(4.1)\\ &\leq 4|n_x| + |h(q(|x|)| + O(1) &(4.2) \end{aligned}$$

Para finalizar, demostremos que para casi todo $x, n_x \leq 2|h(q(|x|)| + O(1)$. Como $C(x) \leq |q_x| + 2n_x + O(1)$, esta cota superior de n_x implicará que

$$\begin{aligned} C(x) &\leq |q_x| + |h(|q_x|)| \\ &= C^A(x) + h(C^A(x)) \end{aligned}$$

para casi todo x, tal como lo deseabamos. Luego, veamos que $n_x \leq 2|h(|q_x|)| + O(1)$ para casi todo x. Existe una constante N tal que para todo $n \geq N$, $8|n| \leq n$. Sabemos que para casi todo x, q_x satisface $|h(|q_x|)| \geq N$. Supongamos que x tiene esta propiedad. Luego $n_x \leq |h(|q_x|)|$ ó $4|n_x/2|$. En éste último caso $n_x - 4|n_x| \geq n_x/2$ y por (2), $n_x/2 \leq |h(|q_x|)| + O(1)$. Con lo cual en ambos casos tenemos que $n_x \leq 2|h(|q_x|)| + O(1)$. □

Lema 4.3.2. *Para todo $x \in \{0,1\}$ y $d \in \mathbb{N}$,*

$$|\{y : C(x,y) \leq C(x) + d\}| = O(d^4 2^d).$$

Demostración. Chaitin [3] ha demostrado que vale

$$\forall d, n \in \mathbb{N} |\{\sigma : |\sigma| = n \wedge C(\sigma) \leq C(n) + d\}| = O(2^d).$$

Sea c tal que $\forall x C(x) \leq str^{-1}(x) + c$. Consideremos la función parcial recursiva $f(x,y,d)$ que enumera a todas las cadenas z tales que $C(z) \leq str^{-1}(x)+d+c$ hasta hallar $z = y$. Si z fue la i-ésima cadena en aparecer en la enumeración, luego $f(x,y,d) = str^{-1}(x)+d+c+1$. Observemos que siempre es posible escribir a $f(x,y,d)$ de esta manera debido a que hay como mucho $2^{str^{-1}(x)+d+c+1}$ tales cadenas z. Si no existe tal z entonces $f(x,y,z) \uparrow$. Sean x y d dados. Consideremos y tal que $C(x,y) \leq C(x) + d$. Como $C(x,y) \leq str^{-1}(x) + c$ entonces $f(x,y,d) \downarrow$ y vale

$$\begin{aligned} C(f(x,y,d)) &\leq C(x,y) + 2|d| + O(1) \\ &\leq C(x) + d + 2|d| + O(1) \\ &\leq C(str^{-1}(x) + d + c + 1) + d + 4|d| + O(1). \end{aligned}$$

La última desigualdad vale debido al hecho de que podemos computar la cadena x a partir de los números $str^{-1}(x)+d+c+1$ y d. Sea $n = str^{-1}(x)+d+c+1$ y $d' = d+4|d|+O(1)$. Para x y d fijos, la función: $y \mapsto f(x,y,d)$ resulta inyectiva con lo cual

$$\begin{aligned} |\{y : C(x,y) \leq C(x) + d\}| &\leq |\{\sigma : |\sigma| = n \ \wedge C(\sigma) \leq C(n) + d'\}| \\ &= O(d^{d'}) = O(d^4 2^2). \end{aligned}$$

□

Teorema 4.3.3. *Son equivalentes:*

(a) A es strongly jump-traceable;

(b) Para cada función de orden h y casi todo x, $C(x) \leq C^A(x) + h(C^A(x))$

Demostración. Para cualquier función f, sea $\hat{f}(y) = y + f(y)$ para todo y. $(a) \Rightarrow (b)$. Sea h_0 una función de orden dada. Es suficiente demostrar que $C(x) \leq C^A(x) + h(C^A(x)) + O(1)$ para casi todo x, con $h = \lfloor h_0/2 \rfloor$. Sea α una función de reducción tal que $J^A(\alpha(x)) = U^A(str(x))$. Sea T una traza para J^A con cota g tal que $g(\alpha(x)) \leq h(|str(x)|)$. Sea $m \in \mathbb{N}$ tal que $U^A(str(m)) = y$ y $|str(m)| = C^A(y)$. Como $y \in T_{\alpha(m)}$, podemos codificar y con m y un número menor o igual a $g(\alpha(m))$ (el cual representa el lugar ($\leq g(\alpha(m))$) dentro de la enumeración de $T_{\alpha(m)}$ en la cual y es enumerado), utilizando como mucho la siguiente cantidad de bits

$$|str(m)| + g(\alpha(m)) \leq C^A(y) + h(C^A(y)).$$

Luego $\forall y, y \leq h(C^A(y)) + O(1)$.

$(b) \Rightarrow (a)$. Puesto que hay como mucho $2^n - 1$ programas de largo menor a $n, \forall n \exists x[|x| = n \wedge n \leq C(x)]$. Sea c una constante tal que

$$\forall x[J^A(|x|) \downarrow \Rightarrow C^A(x, J^A(|x|)) \leq |x| + x].$$

Esta última desigualdad vale debido a que, dado x podemos computar $J^A(|x|)$ relativo a A.

Sea h una función de orden arbitraria, demostremos entonces que A es jump-traceable vía h. Definamos la función de orden g de manera tal que para todo $e, 3^{g(e+c)} \leq h(e)$. Por hipótesis, para casi todo x, si $J^A(|x|) \downarrow$ entonces

$$\begin{aligned} C(x,y) &\leq \hat{g}(C^A(x, J^A(|x|))) \\ &\leq |x| + \hat{g}(|x| + c) + c. \end{aligned}$$

Definimos la siguiente traza

$$T_e = \{y : \forall x[|x| = e \Rightarrow C(x,y) \leq e + g(e+c) + c]\}.$$

Resulta claro que para casi todo e, si $J^A(e) \downarrow$ luego $J^A \in T_e$, puesto que dado x tal que $|x| = e$, tenemos que $C(x, J^A(x)) \leq e + g(e+c) + c$. Para verificar que para casi todo e, $|T_e| \leq h(e)$, supongamos que $y \in T_e$. Tomemos $x, |x| = e$ y $C(x) \geq e$. Luego

$$\begin{aligned} C(x,y) &\leq e + g(e+c) + c \\ &\leq C(x) + g(e+c) + c. \end{aligned}$$

Por el Lema 4.3.2, para casi todo e hay como mucho $3^{g(e+c)} \leq h(e)$ tales y en T_e. □

En [25] se demostró que existe un conjunto super-low que no es jump-traceable (de hecho este conjunto puede pedirse Martin-Löf aleatorio) Sin embargo el Teorema 4.3.1 y el Teorema 4.3.3 nos permiten concluir que la version fuerte de super-lowness implica strong jump-traceability.

Corolario 4.3.4. *Si A' es well-aproximable entonces A es strongly jump-traceable.*

Capítulo 5

K-Trivialidad

En este último capítulo introduciremos una de las nociones fundamentales de la teoría, una clase de conjuntos débiles desde el punto de vista de la aleatoriedad que llamaremos K-triviales. Pero antes de definir esta clase de conjuntos definiremos sus antecesores naturales, los conjuntos C-triviales.

5.1. C-Trivialidad

Diremos que un conjunto A es C-Trivial si $\exists b\ \forall n C(A \upharpoonright n) \leq C(n) + b$. Un resultado de Chaitin [4] nos dice que los conjuntos computables son exactamente los conjuntos C-Triviales. Antes de demostrar este resultado veremos un lema que se utilizará en la demostración y que resulta de interés propio.

Lema 5.1.1. *Dada una máquina M, podemos hallar de manera efectiva $d \in \mathbb{N}$ tal que para todo x, b,*

$$\#\{\sigma : M(\sigma) = x \ \&\ |\sigma| \leq C(x) + b\} < 2^{b+2\log b+d+5} = O(b^2 2^b) \tag{5.1}$$

Demostración. La idea detrás de la demostración es que si existen demasiadas M-descripciones cortas de x, utilizando un listado de estas descripciones podemos construir una $\mathbb{V}$-descripción de x más chica que $C(x)$, lo cual constituye una contradicción.

Recordemos de la página 5 que $str(b)$ es la cadena identificada con b que tiene longitud $\log(b+1)$. Sea $\hat{b} = 0^{|str(b)|}1str(b)$ con lo cual $|\hat{b}| \leq 2\log b+3$. Definimos una máquina R. Por el teorema 2.2.3 (con un parámetro para M) podemos asumir que poseemos una constante de codificación $d > 0$ para R, es decir, $\Phi_d = R$.

La máquina R se construirá de la siguiente manera: para cada b y cada $m > 2\log b+d+4$ y cada x, si hay $2^{b+2\log b+d+5}$ cadenas σ de largo como mucho $m + b$ tales que $M(\sigma) = x$, sea $R(\hat{b}\rho) = x$ para el ρ más a la izquierda de largo $m - 2\log b - d - 4$ de manera que $\hat{b}\rho$ no se encuentre aún en el dominio de R. Notemos que $|\hat{b}\rho| < m - d$.

La definición de R es consistente: para cada b, m hay $2^{m+b+1} - 1$ cadenas de largo como mucho $m+b$, con lo cual como mucho $2^{m+b+1}/2^{b+2\log b+d+5} = 2^{m-2\log b-d-4}$ cadenas x tienen suficientes número de M-descripciones para obtener una R-descripción $\hat{b}\rho$.

Supongamos que (5.1) falla para b, x y sea $m = C(x)$. Luego $2^{m+b+1} \geq 2^{b+2\log b+d+5}$, con lo cual $m \geq 2\log b + d + 4$, asegurándonos entonces de que $C_R(x) < m - d$ con lo cual $C(x) < m$, resultando esto último una contradicción. □

Teorema 5.1.2. *1. Para cada b como mucho $O(b^2 2^d)$ conjuntos son C-triviales con constante b.*

2. Cada conjunto C-trivial es computable.

1) Aplicaremos el lema para contar las cadenas z de tamaño n tales que $C(z) \leq C(n)+b$. Consideremos la máquina M dada por $M(\sigma) \simeq |\mathbb{V}(\sigma)|$. Cada $\mathbb{V}$-descripción mínima de un z es una M-descripción mínima de n de tamaño como mucho $C(n) + b$, con lo cual por el lema anterior, hay como mucho r, donde $r = O(b^2 2^b)$ es independiente de n.

Consideremos el siguiente árbol

$$T_b^C = \{z : \forall u \leq |z| C(z \upharpoonright u) \leq C(u) + b\},$$

que contiene como mucho r cadenas de cada largo n. Cada conjunto que es C-trivial para b es un camino de T_b^C, con lo cual hay como mucho r de tales conjuntos.

2) Sea A C-trivial via b. El árbol T_b^C solo no nos sirve para determinar si A es computable pues es Δ_2^0. Consideraremos en cambio el árbol c.e. $T_b^C \subseteq \tilde{S}_b$ definido como $\{z : \forall u \leq |z|[C(z \upharpoonright u) \leq 1 + log(u+1) + b]\}$. Para casi todo i, existe una cadena y de largo i tal que $C(y) = |y| + 1$ con lo cual existe un número u, $2^i - 1 \leq u < 2^{i+1} - 1$ tal que $C(u) = 1 + \log(u + 1)$. Sea

$$S_b = \{z : \exists w \succeq z[|w| = 2|z| \ \& \ w \in \tilde{S}_b]\},$$

entonces hay como mucho r cadenas en S_b en casi todo nivel $k = 2^i - 1$ de S_b. El conjunto A es un camino en S_b con lo cual existe $z \prec A$ tal que para cada $k > |z|$ de la forma $2^i - 1$, $A \upharpoonright k$ es la única cadena en S_b extendiendo a z. Como el árbol S_b es c.e. podemos enumerarlo hasta que esta cadena aparece. A entonces resulta computable.

5.2. Conjuntos K-Triviales

Definición 5.2.1. A se dirá K-Trivial via b si $K(A \upharpoonright n) \leq K(n) + b$ para todo n.
Llamaremos $\mathcal{K}$ a la clase de conjuntos K-Triviales.

Hecho 5.2.1. *Todo conjunto computable A es K-Trivial.*

Demostración. Basta tomar el conjunto $W = \{\langle n+1, A \upharpoonright n\rangle : n \in \mathbb{N}\}$ como A es computable el conjunto W resulta c.e. y es claro que los axiomas constituyen un conjunto de Kraft-Chaitin, con lo cual para todo n vale que $K(A \upharpoonright n) \leq K(n) + b$. □

5.2.1. Construyendo un conjunto K-Trivial no computable

Nuestra intención es construir un conjunto A que sea K-Trivialy no computable. Dada la definición de K-trivialidad resulta natural enumerar un conjunto de axiomas W que satisfagan las condiciones del teorema de Kraft-Chaitin, esto es, para cada $n \in \mathbb{N}$, $\langle K(n)+1, A \upharpoonright n\rangle \in W$. El primer incoveniente que surge con este intento es que tanto $K(n)$ como $A \upharpoonright n$ nos son desconocidos, con lo cual será necesario trabajar con aproximaciones de estos conjuntos. Si $K_t(w) \leq K_{t-1}(w)$ entonces ingresamos el axioma $\langle K_t(w)+1, A_t \upharpoonright w\rangle$ en W. Más aún, si $x < t$ es minimal tal que $A_{t-1}(x) \neq A_t(x)$, luego para cada w, $x < w \leq t$ ponemos en W el axioma $\langle K_t(w)+1, A_t \upharpoonright w\rangle$. En este caso, los axiomas para descripciones de $A_{t-1} \upharpoonright w$ que enumeramos previamente han sido "desperdiciados". En consecuencia cada cambio en $A(x)$ conlleva un *costo*, el peso desperdiciado en la descripción de la cadena $A_t \upharpoonright w, x < w \leq t$. Supongamos que enumeramos el axioma $\langle K_s(w)+1, A_s \upharpoonright w\rangle$ en W en la etapa $s < t$, añadiendo con ello el peso de $2^{-(K_s(w)+1)}$ a W. Como $2^{-K_s(y)} < 2^{-K_t(y)}$, el costo de cambiar $A(x)$ es como mucho

$$c(x,t) = \frac{1}{2}\sum_{x<y\leq t} 2^{-K_t(y)}.$$

Notemos que $c(x,t)$ es no decreciente en t, $\lim_t c(x,t) \leq \frac{1}{2}$ para cada x y $\lim_x \lim_t c(x,t) = 0$. La noción de costo resulta fructífera y se generaliza de la siguiente manera

Funciones de costo

Definición 5.2.2. Una función de costo es una función computable

$$c : \mathbb{N} \times \mathbb{N} \to \{x \in \mathbb{Q}_2 : x \geqslant 0\}.$$

Diremos que c satisface la *condición del límite* si $lim_x sup_{s>x} c(x,s) = 0$, esto es, $\forall e \forall^{\infty} x \forall s > x[c(x,s) \leq 2^{-e}]$.
Diremos que c es *monótona* si $c(x+1,s) \leq c(x,s) \leq c(x,s+1)$ para cada $x < s$, es decir $c(x,s)$ no disminuye cuando alargamos el intervealo $[x,s)$.

En lo que sigue definiremos los valores de la función de costo $c(x,s)$ para $x < s$, asumiendo que $c(x,s) = 0$ si $x \geqslant s$.

Definición 5.2.3. La función de costo estándar $c_{\mathcal{K}}$ está dada por

$$c_{\mathcal{K}}(x,s) = \sum_{x<w\leq s} 2^{-K_s(w)}.$$

Lema 5.2.4. *$c_{\mathcal{K}}$ es monótona y satisface la condicion del limite.*

Demostración. Que $c_{\mathcal{K}}$ es monótona resulta inmediato. Para ver que satisface la condición del límite dado $e \in \mathbb{N}$, como $\sum_w 2^{-K(w)} \leq 1$, existe un x_0 tal que $\sum_{w\geqslant x_0} 2^{-K(w)} \leq 2^{-e}$. Luego $c_{\mathcal{K}}(x,s) \leq 2^{-e}$ para todo $x \geqslant x_0$ y todo $s > x$. □

Definición 5.2.5. Diremos que una aproximación computable $(A_s)_{s\in\mathbb{N}}$ de un conjunto A que es Δ^0_2 *obedece* a la función de costo c si

$$S = \sum_{x,s} c(x,s)[\![x < s \ \&\ \mu x A_{s-1}(x) \neq A_s(x)]\!] < \infty \tag{5.2}$$

Si la aproximación Δ^0_2 del conjunto A resulta clara del contexto, diremos que el conjunto A obedece la función de costo.

Podemos pensar en las funciones de costo como descripciones de clases de conjuntos Δ^0_2, es decir aquellos conjuntos cuyas aproximaciones obedecen cierta función de costo. Por ejemplo, la función de costo estándar describe a los conjuntos K-Triviales.

Teorema 5.2.6. *Sea c una función de costo que cumple la condición del límite. Entonces existe un conjunto A que es promptly-simple, con una enumeración $(A_s)_{s\in\mathbb{N}}$ que obedece c, más aún, $S \leq \frac{1}{2}$ en (5.2).*

Demostración. Para que sea promplty simple pediremos que se cumplan las siguientes condiciones

$$PS_e : |W_e| = \infty \Rightarrow \exists s \exists x [x \in W_{e,\text{at s}} x \in A_s]$$

(donde $W_{e,\text{at s}} = W_{e,s} - W_{e,s-1}$). Definimos entonces una enumeración computable $(A_s)_{s\in\mathbb{N}}$ de la siguiente manera: Sea $A_0 = \emptyset$. En la etapa $s > 0$, para cada $e < s$ si el requerimiento PS_e no ha sido cumplido hasta el momento y existe $x > 2e$ tal que $x \in W_{e,s}$ y $c(x,s) \leq 2^{-e}$, entonces ingresamos x a A_s y declarmos PS_e satisfecho. La enumeración computable $(A_s)_{s\in\mathbb{N}}$ cumple la condición de costo puesto que en cada etapa ingresamos únicamente un elemento y el costo de este elemento es 2^{-e} luego la suma S de (5.2) se encuentra acotada por $\sum_e 2^{-e} = 2$.

Si W_e es infinito entonces siempre existe un $x < 2e$ en W_e tal que $c(x,s) \leq 2^{-e}$ para $s > x$ puesto que c cumple la condición del límite. Enumeramos tal x en A en la etapa $s > x$ cuando x aparece en W_e en caso de que la condición PS_e no se haya alcanzado aún en la etapa s. Luego A resulta un conjunto promplty simple. Con el mismo razonamiento podemos modificar nuestra construcción para que el costo en el que incurre cada requerimiento sea como mucho 2^{-e-2} de esta manera nos aseguramos que $S \leq \frac{1}{2}$. □

Teorema 5.2.7. *Supongamos que una aproximación computable $(A_s)_{s\in\mathbb{N}}$ de un conjunto A obedece la función de costo estándar $c_{\mathcal{K}}(x,s) = \sum_{x<w\leq s} 2^{-K_s(w)}$. Entonces A es K-Trivial.*

Demostración. Por hipótesis el costo total S de (5.2) es finito. Supongamos en principio que $S \leq 1$. Enumeramos axiomas de Kraft-Chaitin W en la etapa s de la siguiente manera: pondremos el axioma $\langle K_s(w)+1, A_s \upharpoonright w\rangle$ en W cada vez que $w \leq s$ y

(a) $K_s(w) < K_{s-1}(w)$

(b) $K_s(w) < \infty$ & $A_{s-1} \upharpoonright w \neq A_s \upharpoonright w$.

Los axiomas ingresados por a) contribuyen como mucho $\frac{\Omega}{2}$ puesto que

$$\sum_{w<s} 2^{-K_s(w)+1} \leq \frac{1}{2} \sum_{w<s} 2^{-K_s(w)} \leq \frac{\Omega}{2}$$

Supongamos ahora que el axioma $\langle K_s(w)+1, A \upharpoonright w \rangle$ ha sido enumerado por causa de (b). Luego $w > x$ donde x es el mínimo tal que $A_s(x) \neq A_{s-1}(x)$, luego el término $2^{-K_s(w)}$ tiene una ocurrencia en la suma $c_{\mathcal{K}}(x,s)$ y por lo tanto en la suma S. Si asumimos que $S \leq 1$, el aporte de este axioma es de como mucho $\frac{1}{2}$. Por lo tanto W es un conjunto de Kraft-Chaitin.

Sea M_d la máquina obtenida por el teorema de Kraft-Chaitin con los axiomas W. Afirmamos que $K(A \upharpoonright w) \leq K(w) + d + 1$ para cada w. Dado w, sea s tal que $s = 0$ ó $A_s(w) \neq A_{s-1}(w)$. Si $s > 0$ luego el axioma en b) produjo $K_u(A \upharpoonright w) \leq K_s(w) + d + 1$ para algún $u > s$. Si $K_s(w) = K(w)$, entonces hemos arribado a la conclusión deseada. En otro caso la desigualdad ha sido producida por un requerimiento en a) en la etapa más grande $t > s$ tal que $K_t(w) \leq K_{t-1}(w) + d + 1$ (esto incluye el caso $s = 0$ pues $K_0(w) = \infty$) □

Como vimos en el Lema 5.2.4 $c_{\mathcal{K}}$ satisface la condición del límite, esto último junto al Teorema 5.2.6 nos da la siguiente

Proposición 5.2.8. *Existe un conjunto A que es promplty-simple y K-Trivial.*

5.3. Subclases de los conjuntos K-triviales

De la misma forma en que se definieron los tests de Martin-Löf, ver página 22, pueden definirse los *tests de Schnorr*, estos últimos tienen como requerimiento extra que $\mu(U_n) = 2^{-n}$. Se define de manera análoga a la 1-aleatoriedad el concepto de aleatoriedad de Schnorr, con lo cual pueden considerarse entonces los conjuntos bajos para aleatoriedad de Schnorr.

Terwijn y Zambela [35] definieron un conjunto A como *computablemente trazable* si existe una cota computable p tal que para cada $f \leq_T A$, existe una traza computable T con cota p que aproxima los valores de f, es decir $f(n) \in T_n$. Lograron demostrar que A es bajo para aleatoriedad de Schnorr sii A es computablemente trazable. Es decir, lograron una caracterización de un concepto que involucra aleatoriedad utilizando únicamente conceptos de la teoría de la computabilidad.

Por otro lado, se ha demostrado que el concepto de ser K-trivial posee varias equivalencias, pero ninguna de éstas pertenece estrictamente a la teoría de la computabilidad, con lo cual resulta sensato preguntarse si existe una caracterización análoga a la de los conjuntos Low para Schnorr aleatorio. Una tal caracterización podría permitirnos, por ejemplo, hallar demostraciones más sencillas de las equivalencias mencionadas con anterioridad.

Con el objeto de hallar una caracterización en términos de la teoría de la computabilidad de los conjuntos K-triviales se introdujeron los conjuntos strongly jump traceables y superlow tratados en el capitulo anterior. Se tenía la esperanza de que estas propiedades

ayudasen a encontrar la buscada caracterización, sin embargo Cholak, Downey y Greenberg [5] demostraron que existen conjuntos K-triviales y c.e. que no son strongly jump traceable. Mediante el método de promoción de cajas (*box promotion*) lograron demostrar que los conjuntos strongly jump traceable y c.e. son una clase propia de los conjuntos K-triviales.

5.3.1. Un conjunto K-trivial y c.e. que no es strongly jump-traceable

En el teorema 5.2.6 construimos un conjunto promplty simple dada una función de costo arbitraria que cumple la condición del límite. Si consideramos la función de costo estándar $c_{\mathcal{K}}$ descubrimos que ésta nos permite la construcción de un conjunto A K-trivial y c.e. que no es strongly jump traceable. La idea general de la demostración, como es usual en este tipo de construcciones, consiste en considerar ciertas estrategias R_e con $e \in \mathbb{N}$ que hacen que A no sea strongly jump traceable. Más precisamente logramos demostrar que existe un conjunto con las características mencionadas anteriormente con la propiedad de poseer un funcional de Turing Ψ de manera tal que no existe una traza c.e. para Ψ^A con una cota en el orden de $\log\log z$

Teorema 5.3.1. *Existe un conjunto A c.e. y K-trivial que no es strongly jump traceable. De hecho, existe un funcional de Turing Ψ, tal que no existe una traza para Ψ^A con cota $\lambda z.\, \text{máx}(1, \lfloor \frac{1}{2} \log\log z \rfloor)$.*

Detalles de la demostración. Construiremos una enumeración computable $(A_s)_{s\in\mathbb{N}}$ de un conjunto A que obedece la función de costo estándar $c_{\mathcal{K}}$, con lo cual por el teorema 5.2.7, A será K-trivial.

Sea $I_1, I_2, \ldots$ una sucesión de intervalos disjuntos de $\mathbb{N}$ tales que $|I_e| = 2^{e2^e}$. Sea h una función de orden tal que $h(z) = e$ si $z \in I_e$. Notemos que $\frac{1}{2}\log\log z \leq h(z)$ para cada z puesto que $\sum_{1\leq i\leq e} 2^{i2^i} = \text{mín}\, I_e \leq 2^{2^{2e}}$ para cada $e > 0$. Sea $(S^e_z)_{z\in\mathbb{N}}$ la e-ésima traza c.e. con cota h en un listado uniformemente c.e. de todas las trazas.

Construiremos un funcional de Turing Ψ de manera tal que para cada $e > 0$ existe un z tal que $\Phi^A(z) \notin S^e_z$ para algún $z \in I_e$. Como $|S^e_z| \leq e$ resulta suficiente tener a disposición $e+1$ números para enumerar en A antes de definir $\Psi^A(z)$ de manera definitiva con un uso suficientemente grande como para que no pertenezca a la traza.

El objetivo de nuestra construcción es lograr lo anterior y al mismo tiempo obedecer la función de costo estándar $c_{\mathcal{K}}$ para que nuestro conjunto sea K-trivial. Con el objeto de motivar la idea general de la demostración reformularemos la demostración de la existencia de un conjunto K-trivial y no computable en términos de procedimientos. Tenemos como requerimiento que $A \neq \Phi_e$ para todo e. El procedimiento $P^e(b)$ para este requerimiento puede incurrir en un costo fijo de $1/b$. Intentará, por lo tanto, enumerar a w en A en la etapa s cuando $\Phi_{e,s}(w) = 0$ para lograr la diagonalización, es decir que $\Phi_e(w) = A(w)$ falle. Para ello busca un número w tal que $c_{\mathcal{K}}(w, s) \leq 1/b$ en la etapa s cuando desea ingresar w a A.

Procedimiento $P^e(b)$ $(e > 0, b \in \mathbb{N})$

1. Sea w el número de etapa actual.

2. *ESPERAR* una etapa s tal que $\Phi_{e,s}(w) = 0$ Si s es hallado, entonces ingresamos w a A_s. Si $c_{\mathcal{K}}(w,t) \leq 1/b$ en la etapa t durante la espera, entonces *GOTO* 1. Diremos en este caso que el procedimiento P_e ha sido *reseteado*.

Podemos pensar que cuando un procedimiento ha sido reseteado uno de los intentos de lograr nuestro objetivo ha fallado, razón por la cual debemos volver a comenzar en otras condiciones.

Como $\sum_n 2^{-K(n)} \leq 1$ el procedimiento $P^e(b)$ es reseteado como mucho b veces, con lo cual eventualmente w se estabiliza. En la etapa e de la construcción inicializamos el procedimiento $P^e(2^e)$ como cada uno de los procedimientos enumera un número una única vez entonces el costo total de los cambios es finito.

En nuestro caso particular tenemos como requerimiento que para cada e, S_z^e no sea una traza de Ψ^A, para ello necesitamos disponer, en el peor de los casos, de $e+1$ números para enumerar en A. Utilizaremos cada uno de estos números para llenar un lugar de la traza que posee e lugares, de esta manera nos aseguramos de que el lugar $e+1$-ésimo no pertenezca a la traza. La discusión previa nos sugiere la existencia de subprocedimientos dentro de nuestro procedimiento P^e, es decir, tenemos $P_j^e, e \geq j \geq 0$. La estrategia comienza con P_e^e que llama al procedimiento P_{e-1}^e hasta llegar a P_0^e el cual escoge un $z \in I_e$ y define $\Psi^A(z)$ por primera vez. En cada ocasión que un procedimiento halla un valor de nuestro funcional Ψ^A en la traza S_z^e entonces hemos logrado una victoria parcial, el subprocedimiento informa de esto a su procedimiento padre y éste redefine nuestro funcional con un valor más grande. De manera similar a la construcción de un conjunto K-trivial, en cada momento permitimos que un subprocedimiento incurra en un costo específico. El costo del procedimiento P_j^e será de $1/b_{e,j}$ y estos costos podrán computarse con antelación.

Durante el transcurso de un subprocedimiento puede ocurrir que una estrategia particular resulte fallida, es decir que el subprocedimiento sea *reseteado*. Esta estrategia trunca puede que incurra en un cierto costo que puede llegar a atentar contra nuestros deseos por lo cual debemos acotar de alguna manera estos *costos residuales*. La solución consiste en obligar a un procedimiento hijo a tener un capital menor al del procedimiento padre, es decir si el capital del procedimiento padre es de $1/b$ entonces el procedimiento hijo podrá gastar únicamente $1/b^2$. Veremos que un procedimiento P_j^e puede ser reseteado una cantidad finita de veces. Con las restricciones de costo mencionadas con anterioridad veremos que podemos permitirnos incurrir en tales gastos. La clave se encuentra por lo tanto en la elección particular de costo permitido por subprocedimiento y por el largo del intervalo I_e escogido de manera tal que siempre resulta posible elegir un nuevo número $z \in I_e$. Podemos pensar en cada número del intervalo I_e como un posible testigo de que $\Psi^A(z) \notin S_z^e$, como algunos de estos testigos pueden fallar necesitamos suficientes. Nuestra estrategia triunfa debido a que podemos determinar *a priori* la cantidad de testigos necesaria.

En lo que sigue $e > 0$ y llamaremos a $P_e^e(2^e)$ en la etapa e.

Procedimiento $P_0^e(b)$ (En este nivel el b resulta accesorio y tiene como utilidad simplificar la notación)

1. En la etapa actual s, sea z el mínimo número utilizado en I_e. Defino $y = \Psi^A(z)[s] = s$ con uso s.

2. *ESPERAR* a una etapa $r > s$ tal que $y \in S^e_{z,r}$. Mientras tanto, si $A \upharpoonright y$ cambia, entonces redefinimos $\Psi^A(z) = y$ con el mismo uso y. Si hallamos tal r entonces *RETORNAMOS* z

Procedimiento $P^e_j(b)(b, j \in \mathbb{N}, e > 0, j > 0)$

1. Sea w la etapa actual. *Llamar* al procedimiento $P^e_{j-1}(b^2)$.

2. *ESPERAR* una etapa s en la cual el procedimiento hijo P^e_{j-1} *retorne* un número z. Si una tal etapa s es hallada entonces ingresamos w a A_s. Como el w es menor al uso de $\Psi^A(z)[s-1]$ entonces $\Psi^A(z)$ queda indefinida. Entonces sea $y = s$, redefinimos $\Psi^A(z)[s] = y$ con uso y.

   Puede ocurrir que durante nuestra espera en la etapa t se dé que $c_{\mathcal{K}}(w, t) > 1/b$, si esto sucede, en el inicio de la etapa t cancelamos entonces el procedimiento hijo P^e_{j-1} y toda su descendencia y volvemos luego al primer paso. En este caso decimos que P^e_j ha sido *reseteado*. Notemos que han sido cancelados los subprocedimientos de P^e_j y que todavía no se ha incurrido en ningún costo.

3. *ESPERAR* una etapa $r > s$ tal que $y \in S^e_{z,r}$. Entretanto, si $A_{t-1} \upharpoonright y \neq A_t \upharpoonright y$ en la etapa actual t entonces redefinimos $\Psi^A(z) = y$ con el mismo uso que antes. Si un tal r es hallado entonces *RETORNAMOS* z.

Algunas observaciones resultan pertinentes. El objetivo del procedimiento P^e es hallar un z tal que $\Phi^A(z) \notin S^e_z$. Para ello dispone de subprocedimientos que se encargan de hallar e números dentro del intervalo I_e y busca definir provisoriamente nuestro funcional con usos que pertenezcan a la traza, si logra hallar e de éstos números se decide finalmente por un uso mayor que todos los utilizados con lo cual se asegura de que $\Phi^A(z) \notin S^e_z$. Sin embargo esto no necesariamente sucede siempre, es decir, puede que un procedimiento se quede esperando que el valor definido $y \in S^e_z$ y esto no suceda nunca con lo cual habríamos logrado nuestro objetivo pero de otra manera, como no se puede tener certeza de ésto, la estrategia en este caso es mantener el valor incluso cuando otros requerimientos cambian la aproximación de A por debajo del uso hasta que $y \in S^e_z$ en caso de que esto suceda.

Hecho 5.3.1. *Un procedimiento $P^e_j(b)$ no es reseteado más de b veces.*

Demostración. Supongamos que el procedimiento $P^e_j(b)$ comienza en la etapa w_0 y es reseteado en las etapas $w_1 < \ldots < w_k$. Entonces $k/b < \sum_{0 \leq i \leq k} c_k(w_i, w_{i+1}) \leq \Omega \leq 1$. Luego $k < b$. □

Los números $b_{e,j}(j \leq e)$ están dados por la siguiente recursión: $b_{e,e} = 2^e$ y $b_{e,j-1} = b^2_{e,j}$ si $j > 0$; en otras palabras tenemos que $b_{e,j} = 2^{e2^{e-j}}$. Notemos que sólo llamamos a un procedimiento $P^e_j(b)$ para $b = b_{e,j}$.

Hecho 5.3.2. *Para cada $j \leq e$ el procedimiento $P_j^e(b)$ es llamado como mucho $b_{e,j}2^{-e}$ veces.En particular, como $|I_e| = b_{e,0}$ un procedimiento P_0^e siempre tiene a disposición un $z \in I_e$ para elegir.*

Demostración. Utilizaremos inducción hacia atrás para $j \leq e$. El procedimiento P_e^e es llamado únicamente una vez. Supongamos que nuestra afirmación vale para $j > 0$. Como cada procedimiento P_j^e es llamado con el parámetro $b_{e,j}$ cada procedimiento es reseteado menos de $b_{e,j}$ veces. Por lo tanto P_{j-1}^e es llamado como mucho $b_{e,j}^2 2^{-e} = b_{e,j-1}2^{-e}$ veces. □

Hecho 5.3.3. *Existe un $z \in I_e$ tal que $\Phi^A(z)\downarrow \notin S_z^e$.*

Demostración. Sea $z \in I_e$ el último número elegido en un procedimiento P_0^e. Ningún procedimiento $P_j^e, j > 0$ es reseteado luego de la etapa s en que z es elegido, puesto que si esto sucediese entonces z no sería el último número elegido ya que volveríamos a correr un procedimiento P_0^e. Supongamos que $r \leq e+1$ es el número más grande tal que para $j < r$ el procedimiento P_j^e en la etapa s retorna en la etapa s_j. Sea $y_j = \Psi^A(z)[s_j]$. Luego $y_0 < \ldots < y_{r-1}$ y $y_j \in S_z^e$ con lo cual $r \leq e$ puesto que $|S_z^e| \leq e$. Por la acción en 3) o en 2) si $r = 0, y = \Psi^A(z)\downarrow$. Más aún, $y \notin S_z^e$ puesto que el procedimiento P_r^e no retorna. □

Hecho 5.3.4. *A es K-trivial.*

Demostración. Por el teorema 5.2.7 bastará demostrar que $(A_s)_{s\in\mathbb{N}}$ obedece la función de costo $c_{\mathcal{K}}$. Si $w \in A_s - A_{s-1}$ esto es producido por 2) a partir de un procedimiento $P_j^e(b_{e,j}), j > 0$. Luego $c_{\mathcal{K}}(w, s) \leq 1/b_{e,j}$ en otro caso el procedimiento se hubiese reseteado en el inicio de la etapa y w jamás hubiese ingresado en A. Por el hecho 5.3.2 P_j^e es llamado como mucho $b_{e,j}2^{-e}$ veces. Como únicamente ingresa un solo elemento a A por llamada del procedemiento luego la suma 5.2 satisface

$$S \leq \sum_{0<e} \sum_{j=1}^{e} b_{e,j}2^{-e}/b_{e,j} \leq \sum_{0<e} e2^{-e} < \infty.$$

□

Una modificación del resultado anterior debida a Keng Meng Ng [21] nos permite probar el siguiente

Teorema 5.3.2. *Para cada función de orden b existe una función de orden h y un conjunto A c.e. que es jump traceable vía b pero no vía h.*

5.3.2. Los conjuntos c.e. y strongly jump-traceables son K-triviales

Definición 5.3.3. Una función de costo monótona c se dirá *benigna* si existe una función computable g tal que para todo racional positivo $\varepsilon, g(\varepsilon)$ acota el tamaño de cualquier colección I de intervalos de números naturales, disjuntos dos a dos tales que para todo $[x, s) \in I$ tenemos que $c(x, s) \geq \varepsilon$.

Por ejemplo, la función de costo estándar $c_{\mathcal{K}}(x,s)$ es benigna: para cualquier $\varepsilon > 0$, cualquier colección I no puede tener una medida mayor a $\frac{1}{\varepsilon}$ debido a que los testigos en la máquina universal para $c_{\mathcal{K}}(x,s) \geq x$ de $[x,s) \in I$ deben ser todos distintos debido a que los intervalos que componen I son todos disjuntos.

Podemos ver también esta definición de la siguiente manera: Sea $\varepsilon > 0$, definimos $y_0^\varepsilon = 0$ y si y_k^ε se encuentra definido y existe un s tal que $c(y_k^\varepsilon, s) \geq \varepsilon$ entonces y_{k+1}^ε es el mínimo s que cumple lo anterior. Si c satisface la condición del límite entonces $\lim_x c(x) = 0$ entonces este proceso debe finalizar en un número finito de pasos. Luego c es *benigna* si y sólo si existe una cota computable en ε de la cantidad de pasos de este proceso. En el siguiente teorema utilizaremos la noción de *traza universal* $\langle S_n \rangle$ para una función de orden p. Sea $\hat{p} = \lfloor \sqrt{p} \rfloor$. Existe una enumeración efectiva $\langle S^1, S^2, S^3, \ldots \rangle$ de las trazas c.e. que se encuentran acotadas por una función de orden p. Sea $S_n = \cup_{e<\hat{p}(n)} S_n^e$. Entonces resulta sencillo ver que $\langle S_n \rangle$ es una traza c.e. que se encuentra acotada por p y para cada función parcial ψ, si ψ tiene una traza c.e. que se encuentra acotada por $\hat{p}$ entonces para casi todo $n \in \text{dom}\psi, \psi(n) \in S_n$ con lo cual podemos decir que S_n *casi traza* ψ.

Notemos que para cada función de orden h' existe una función de orden h tal que para cualquier conjunto X si J^X tiene una traza c.e. acotada por h, entonces cada función parcialmente computable posee una traza acotada por h'. Esto último se debe a la universalidad del salto. Por lo tanto si $\langle T_n \rangle$ es una traza universal para $\tilde{h} = (h')^2$ y J^X tiene una traza c.e. acotada por h entonces cada función X-parcial es casi trazada por $\langle T_n \rangle$. En lo que sigue bastará con definir la función computable $\tilde{h}$ la función h del teorema se sigue de la observación anterior.

Nuestra intención es hallar una función de orden h tal que si A es c.e. y A es h-jump traceable entonces A obedece c, donde c es una función de costo benigna arbitraria. Para ello propondremos con antelación una función de orden $\tilde{h}$ que definirá implícitamente la función h buscada. La función $\tilde{h}$ estará involucrada en la construcción de la enumeración $\langle \widehat{A}_s \rangle$ en tanto que en cada etapa se buscará certificar prefijos de A con distintos niveles de certeza que dependerán de una traza T_z acotada por $\tilde{h}$. Con el objeto de hallar una enumeración de $\langle \widehat{A}_s \rangle$ tomaremos como punto de partida una enumeración c.e. $\langle A_s \rangle$. Si conociésemos a la perfección el comportamiento de esta enumeración resultaría sencillo seleccionar un subconjunto de etapas cuyo resultado final obedeciese a la función de costo c. Como esto no es así tendremos que seleccionar cuidadosamente qué etapas serán parte de nuestra enumeración final de manera tal de poder controlar el costo de los elementos que se agregan. En este sentido podemos decir que la enumeración $\langle \widehat{A}_s \rangle$ es una version *acelerada* de la enumeración $\langle A_s \rangle$.

Como resulta usual en este tipo de construcciones definiremos un función Φ^A A-parcialmente computable que utilizaremos para certificar prefijos de A: para certificar $A_s \upharpoonright u$, definimos en la etapa s, $\Phi_s^A(z) = s$ para ciertos z con uso u, luego diremos que ésta configuración se encuentra certificada en una etapa posterior t si $A_t \upharpoonright u = A_s \upharpoonright u$ y $s \in T_z$ en la etapa t.

Una de las ideas fundamentales de la construcción consiste en tratar el ingreso de cada x a nuestra enumeración $\langle \widehat{A}_s \rangle$ por lotes de costo similar, esto nos permite tratar una cantidad finita de casos de manera uniforme. Estos lotes de costo similar constituirán nuestra estrategia R_n que tratará aquellos x tales que $c(x,s) \geq 2^{-n}$. La estrategia R_n

ignorará aquellos casos para los cuales $c(x,s) \geq 2^{-(n-1)}$ debido a que estos casos serán tratados por la estrategia R_{n+1}. Pondremos un límite a la cantidad de veces que podemos equivocarnos en un prefijo para un determinado costo. Si nos encontramos evaluando un prefijo en la estrategia R_n la cantidad de errores permitidos será n. De esta manera la suma

$$\sum_{s<\mathbb{N}} c(x,s)[\![\mu x(\widehat{A}_s(x) \neq \widehat{A}_{s+1}(x))]\!]$$

se encontrará acotada por $\sum_{n<\mathbb{N}} n2^{-n}$, que es finita, con lo cual la enumeración $\langle \widehat{A}_s \rangle$ obedecerá la función de costo c.

Resulta esencial en este tipo de argumentos la manera en la que un prefijo es certificado en varias etapas. La forma en que la estrategia R_n ordena este proceso es considerando en un inicio un intervalo I_n llamado la *caja* de la estrategia R_n, este intervalo contiene los números z donde se testeará la función Φ^A. A medida que nuestro procedemiento avance iremos subdiviendo la caja inicial en otras de menor tamaño. Llamaremos a estos subintervalos que pertencen a la caja inicial: *meta-cajas*. Avanzaremos de una caja a otra de menor tamaño cuando en una etapa posterior nos percatemos de que un prefijo certifcado es incorrecto. Cada vez que esto suceda Φ^A se indefinirá lo cual nos permitirá redefinirla luego en una meta-caja adecuada. El hecho de que c sea una función de costo benigna, con una cota computable g nos permite fijar de manera anticipada el tamaño de la caja inicial I_n. Como queremos que la estrategia R_n divida su caja inicial en como mucho n meta-cajas, el tamaño de la caja inicial será de $(g(2^{-n}))^{n+1}$.

Teorema 5.3.4. *Para cualquier función de costo benigna c, existe una función de orden h con la siguiente propiedad: si A es un conjunto c.e. tal que J^A tiene una traza c.e. $\{T_e\}_{e\in\mathbb{N}}$ de orden h, entonces A obedece c.*

Demostración. Sean $\langle I_n \rangle_{n\geqslant 1}$ intervalos disjuntos dos a dos y consecutivos de $\mathbb{N}$ tales que $|I_n| = n(g(2^{-n}))^{n+1}$. Para todo $z \in I_n$, sea $\tilde{h}(z) = n$. Luego dividimos cada intervalo I_n en intervalos $I_n^1, I_n^2, \ldots, I_n^n$ cada uno de tamaño $(g(2^{-n}))^{n+1}$. El intervalo I_n^d será utilizado para la $d-$versión de R_n, que denotaremos R_n^d. Tomamos también $B_{n,0}^d = I_n^d$ como la meta-caja inicial para R_n^d.

En cualquier etapa $s, B_{n,s}^d$ será un intervalo de $\mathbb{N}$ cuya medida será una potencia de $g(2^{-n})$. Para $k \in \{1, 2, \ldots, g(2^{-n})\}$ denotaremos el k-ésimo subintevalo de $B_{n,s}^d$ de medida $|B_{n,s}^d|/g(2^{-n})$ como $B_{n,s}^d(k)$.

d-etapas y certificación Como mencionamos con anterioridad, la $d-$versión de nuestra construcción acierta cuando nos dice que para todo $n \geqslant d$, para todo $z \in I_n^d$, si $\Phi^A(z)\downarrow$ luego $\Phi^A(z) \in T_z$. Las $d-$etapas s_i^d serán definidas de manera recursiva: $s_0^d = 0$, dado s_i^d, obtenemos s_{i+1}^d como la etapa s más chica tal que $s > s_i^d$ y para todo $n \in [d,i]$, para todo $z \in I_n^d$, o bien $\Phi^{A_s}(z)\uparrow$ ó $\Phi^{A_s}(z) \in T_{z,s}$. Nos aseguraremos de que la $d-$versión de la construcción sólo haga definiciones de Φ^A en $d-$etapas. Luego, si la $d-$versión adivina de manera correcta, habrá infinitas $d-$etapas.

Para una $d-$etapa $s = s_{i+1}^d$, sea $\overline{s} = s_i^d$, es decir la $d-$etapa previa. Diremos que $A \upharpoonright u$ ha sido *certificado* si

$$A_s \upharpoonright u = A_{\overline{s}} \upharpoonright u.$$

Definición de Φ. Fijemos $d < \mathbb{N}$ y $n \geqslant d$. Sea $i > n$ y $s = s_i^d$. Describiremos la acción de R_n^d en la etapa s. Denotaremos, por simplicidad, a la sucesión $\langle y_k^{2^{-n}} \rangle$, como $\langle y_k^n \rangle : y_0^n = 0$ y si y_k^n se encuentra definido, luego y_{k+1}^n es el mínimo s tal que $c(y_k^n, s) \geqslant 2^{-n}$, en caso de que tal s exista; si esto último no ocurre y_{k+1}^n permanece indefinido. En la etapa s podemos calcular todos los y_k^n tales que $y_k^n \leq s$. El hecho de que nuestra función de costo sea benigna nos dice también que $y_{g(2^{-n})}^n$ no está definida. Nuestro objetivo es que al final de la etapa s, si $k \geqslant 1, y_k^n \leq \bar{s}$ está definida y $A_s \upharpoonright y_k^n$ se encuentra certificado, entonces tendremos $\Phi^{A_s}(z) \downarrow$ con uso y_k^n para todo $z \in B_{n,s}^d(k)$, en cuyo caso diremos que $A_s \upharpoonright y_k^n$ ha sido *testeado* en $B_{n,s}^d(k)$. La hipótesis inductiva de la construcción nos dice que esto vale para tales k al finalizar la etapa $\bar{s}$, mientras que si y_k^n no se encuentra definido en la etapa $\bar{s}$, o está definido pero $A_{\bar{s}} \upharpoonright y_k^n$ no se encuentra certificado, entonces para todo $z \in B_{n,\bar{s}}^d(k)$ tendremos $\Phi^{A_{\bar{s}}}(z) \uparrow$.

Primero, con el objeto de ver si la estrategia R_n^d puede progresar chequeamos si existe un *testigo* $k \geq 1$ tal que $A_{\bar{s}} \upharpoonright y_k^n$ fue testeado en $B_{n,\bar{s}}^d(k)$ y

$$A_s \upharpoonright y_k^n \neq A_{\bar{s}} \upharpoonright y_k^n.$$

En este caso, R_n^d puede *promover* su meta-caja B_n^d: En este caso redefinimos nuestra caja

$$B_{n,s}^d = B_{n,\bar{s}}^d(k),$$

donde k es el menor de los testigos que cumple las condiciones anteriores. Observemos que en este caso, para cada $z \in B_{n,s}^d$ que tenemos, antes de realizar cualquier definicion $\Phi^{A_s}(z) \uparrow$ puesto que en la etapa $\bar{s}$ tenemos $\Phi^{A_{\bar{s}}} \downarrow$ con uso y_k^n. Con lo cual podemos definir $\Phi^{A_s}(z)$ para cada z de manera arbitraria: para todo $l \in [1, k)$, $A_s \upharpoonright y_l^n$ se encuentra certificado, entonces para todo $z \in B_{n,s}^d(l)$ definimos $\Phi_s^A(z) = s$ con uso y_l^n.

Ahora bien, si R_n^d no promociona su meta-caja en la etapa s entonces definimos $B_{n,s}^d = B_{n,\bar{s}}^d$; para todo $k \geq 1$ tal que $A_{\bar{s}} \upharpoonright y_k^n$ fue testeado en la etapa $\bar{s}$, $A_s \upharpoonright y_k^n$ continua certificado y sigue estando testeado en $B_{n,s}^d(k) = B_{n,\bar{s}}^d(k)$. Si existen k tal que $y_k^n \leq s$ y $A_s \upharpoonright y_k^n$ se encuentra certificado pero $A_{\bar{s}} \upharpoonright y_k^n$ no fue testeado en la etapa $\bar{s}$ entonces para todo $z \in B_{n,s}^d(k) = B_{n,\bar{s}}^d(k)$ tenemos $\Phi^{A_{\bar{s}}}(z) \uparrow$, entonces podemos definir $\Phi^{A_s}(z) = s$ con uso y_k^n para todo z.

Con esto finalizamos la construcción. Antes de definir la nueva enumeración $\langle \widehat{A}_s \rangle$ que obedecerá la función de costo c necesitamos comprobar que la construcción es consistente, es decir que puede llevarse a cabo. Resulta sencillo verificar que se cumple la hipótesis inductiva al finalizar la etapa s: si $y_k^n < s$ y $A_s \upharpoonright y_k^n$ se encuentra certificado entonces está testeado en $B_{n,s}^d(k)$, en otro caso para todo $z \in B_{n,s}^d(k)$ tenemos $\Phi^{A_s}(z) \uparrow$. Resulta necesario verificar que cada estrategia R_n^d puede promover siempre su meta-caja B_n^d cuando es requerido, es decir, puede dividir $B_{n,s}^d(k)$ en por lo menos $g(2^{-n})$ subintevalos. Pero esto se sigue del hecho de que la meta-caja inicial $B_{n,0}^d$ es I_n^d y de que vale el siguiente

Lema 5.3.5. *El procedemiento R_n^d no promueve su meta-caja más de n veces.*

Demostración. Sean $r < t$ dos etapas en las que R_n^d promueve su meta-caja. Notemos que para todo $s \geq r$ y para todo $z \in B_{n,s}^d$, si $\Phi^{A_s}(z) \downarrow$ entonces $\Phi^{A_s}(z) \geq r$: cuando $B_{n,r}^d$ es

redefinido en la etapa r tenemos $\Phi^{A_r}(z)$ indefinido, y todas las definiciones ulteriores a r se realizan con el valor de la etapa considerada. Sea k tal que $B^d_{n,t} = B^d_{n,\bar{t}}(k)$. Debido a las condiciones requeridas para que una caja se promueva, tenemos que $A_{\bar{t}} \upharpoonright y^n_k$ se encuentra certificado y testeado, con lo cual $\Phi^{A_{\bar{t}}}(z) \in T_{z,t}$ (donde utilizo la definición de d-etapa). Por lo tanto existe un número $s \in [r, t) \in T_z$ para todo z.

Con un razonamiento análogo podemos ver que si t es la primea etapa en la que B^d_n ha sido promovida entonces existe un número menor a t en T_z para todo $z \in B^d_{n,t}$.

Como las meta-cajas se encuentran encajadas, si B^d_n fuese promovido $n+1$ veces en la $(n+1)$-etapa s tendríamos entonces $n+1$ números en T_z para todo $z \in B^d_{n,s}$. Esto último sería un absurdo puesto que como $B^d_{n,s} \subset I^d_n$, para todo $z \in I^d_n$ tendríamos $n = \tilde{h}(z) \geq |T_z|$. □

Definiremos ahora $\langle \widehat{A_s} \rangle$ y veremos que esta enumeración de A obedece c.

En lo que sigue fijaremos d de manera tal que valga para todo $n \geq d$ y para todo $z \in I^d_n$, si $\Phi^A(z) \downarrow$ entonces $\Phi^A(z) \in T_z$. De esta manera dejaremos de lado en la notación la d, es decir escribiremos R_n en vez de R^d_n, s_i por s^d_i, etcétera.

Definiremos una subsucesión de etapas por recursión. Sea $q(0) = 0$ y dado $q(r)$, definiremos $q(r+1)$ como la mínima etapa s mayor a $q(r)$ en la cual $A_s \upharpoonright q(r)$ se encuentra certificado. Para todo $r < \mathbb{N}$, sea $\widehat{A}_r = A_{q(r+2)} \upharpoonright r$. Para todo r sea x_r el mínimo x tal que $\widehat{A}_{r-1}(x) \neq \widehat{A}_r(x)$, con lo cual $x_r < r$. Sea n_r el único n tal que

$$2^{-n} \leq c(x_r, r) < 2^{-(n-1)}.$$

De esta manera, demostrar que $\langle \widehat{A_r} \rangle$ obedece c resulta equivalente a ver que

$$\sum_r 2^{-n_r} < \infty.$$

Lema 5.3.6. *Para cualquier r, existe una etapa $s \in (q(r+1), (q(r+2)]$ en la cual el procedemiento R_{n_r} promueve su meta-caja.*

Demostración. Sea $n = n_r$ y $x = x_r$. Sea k el mayor número tal que y^n_k se encuentra definido y $y^n_k \leq r$. Tenemos entonces que $x < y^n_k$, en otro caso, por la monotonicidad de c tendríamos que

$$c(y^n_k, r) \geq c(x, r) \geq 2^{-n}$$

lo cual implicaría que y^n_{k+1} está definido y no es mayor que r y habíamos dicho que y^n_k era el mayor número menor o igual que r.

Por lo tanto $y^n_k \leq r \leq q(r)$. La elección de x y el hecho de que $x < y^n_k$ nos dice que

$$A_{q(r+2)} \upharpoonright y^n_k \neq A_{q(r+1)} \upharpoonright y^n_k.$$

Sin embargo por la definición de $q(r+1)$ y el hecho de que $y^n_k \leq q(r)$, $A_{q(r+1)} \upharpoonright y^n_k$ se encuentra certificado, con lo cual está testeado en $B_{n,q(r+1)}(k)$ en la etapa $q(r+1)$.

Obtenemos un cambio en $A_s \upharpoonright y_k^n$ en la etapa $q(r+2)$, con lo cual si s es la menor etapa luego de $q(r+1)$ en la cual $A_s \upharpoonright y_k^n$ no está certificado, entonces $s \leq q(r+2)$ y R_n promueve su meta-caja en la etapa s. □

Se sigue entonces que para todo n

$$\{r : n_r = n\}$$

tiene como mucho n elementos, con lo cual

$$\sum_r 2^{-n_r} \leq \sum_n n2^{-n} < \infty.$$

Esto concluye la demostración del teorema 5.3.4 □

Corolario 5.3.7. *Si A es un conjunto c.e. y strongly jump traceable entonces A es K-trivial*

Demostración. La demostración se sigue del teorema 5.3.4 considerando la función de costo estándar $c_{\mathcal{K}}$, el teorema 5.3.4 nos dice que existe una función de orden h tal que si A es c.e. y J^A tiene una traza c.e. acotada por h entonces A obedece $c_{\mathcal{K}}$. Podemos pedir entonces que J^A tenga una h-traza puesto que nuestra hipótesis nos dice que A es strongly jump traceable y c.e. □

Conjetura

Al momento de finalizar esta tesis no se ha encontrado aún una caracterización de los conjuntos K-triviales que no utilice de alguna manera conceptos de aleatoriedad. Sin embargo, en su búsqueda, se han introducido ideas cuyo impacto es profundo dentro del estudio de la interacción entre aleatoriedad y teoría de la computabilidad. Acaso una investigación pormenorizada de los teoremas 5.3.1 y 5.3.4 arroje luz respecto a posibles direcciones futuras. En este sentido, resulta interesante remarcar el siguiente hecho: En el teorema 5.3.1 hallamos un conjunto A c.e y K-trivial que no es strongly jump traceable, para ello se construyó un funcional Ψ para el cual no es posible hallar una traza con una cota del orden de $\log\log z$. Por otro lado en el teorema 5.3.4 probamos que para cualquier función de costo benigna c existe una función de orden h del orden de $\sqrt{\log z}$ tal que si A es c.e. y J^A tiene una cota c.e. de orden h entonces A obedece c. En ambos casos no se ha usado la definición específica de ser strongly jump traceble sino ciertos ordenes particules. Dicho de otra forma, el pedir una traza para todos los ordenes resulta excesivo a la hora de caracterizar los conjuntos K-triviales, los cuales únicamente requieren poseer una aproximación que obedezca la función de costo estándar $c_{\mathcal{K}}$. Cholak, Downey y Greenberg sugieren buscar una caracterización de los K-triviales que involucre un rango de ordenes comprendido entre los extremos mencionados anteriormente. Sugieren, a su vez, la siguiente

Conjetura 5.3.8. *A es K-trivial sii para todo orden h con $\Sigma_{s\in\mathbb{N}}2^{-h(s)} < \infty$ A es jump-traceable via la función de orden h.*

Bibliografía

[1] Cristian Calude. *Information and Randomness, an Algorithmic Perspective*. Springer-Verlag, Berlin, 1994.

[2] Cristian Calude and Richard J. Coles. Program size complexity of initial segments and domination relation reducibility. In J.Karhümaki, H.Mauer, G.Paŭn un un un, and G.Rozenberg, editors, *Jewels are Forever*, pages 225–237. Springer-Verlag, 1999.

[3] Gregory J. Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM*, 22:329–340, 1975.

[4] Gregory J. Chaitin. Information-theoretical characterizations of recursive infinite strings. *Theoretical Computer Science*, 2:45–48, 1976.

[5] Peter Cholak, Rod Downey, and Noam Greenberg. Strongly jump-traceability I: the computably enumerable case. *Advances in Mathematics*, 217:2045–2074, 2008.

[6] Rod G. Downey, Denis R. Hirschfeldt, André Nies, and Frank Stephan. Trivial reals. *Electronic Notes in Theoretical Computer Science*, 66(1), 2002. Final version in [7].

[7] Rod G. Downey, Denis R. Hirschfeldt, André Nies, and Frank Stephan. Trivial reals. In *Proceedings of the 7th and 8th Asian Logic Conferences*, pages 103–131. World Scientific, River Edge, NJ, 2003.

[8] Santiago Figueira, André Nies, and Frank Stephan. Lowness properties and approximations of the jump. In *12th Workshop on Logic, Language, Information and Computation*, volume 143 of *Electronic Notes in Computer Science*, pages 45–57, 2005. Final version in [9].

[9] Santiago Figueira, André Nies, and Frank Stephan. Lowness properties and approximations of the jump. *Annals of Pure and Applied Logic*, 152(1-3):51–66, 2008.

[10] D.R. Hirschfeldt, A. Nies, and F. Stephan. Using random sets as oracles. *Journal of the London Mathematical Society*, 75(3):610, 2007.

[11] Stephen Cole Kleene. On notations for ordinal numbers. *The Journal of Symbolic Logic*, 3:150–155, 1938.

[12] Andrei N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–17, 1965.

[13] L. G. Kraft. A device for quantizing, grouping, and coding amplitude modulated pulses. M.Sc. Thesis, Massachusetts Institute of Technology, 1949.

[14] A. Kučera and S.A. Terwijn. Lowness for the class of random sets. *Journal of Symbolic Logic*, pages 1396–1402, 1999.

[15] Leonid A. Levin. Some theorems on the algorithmic approach to probability theory and information theory. Dissertation in Mathematics, Moscow, 1971.

[16] Leonid A. Levin. Laws of information conservation (non-growth) and aspects of the foundations of probability theory. *Problems of Information Transmission*, 10(3):206–210, 1974.

[17] Ming Li and Paul M.B. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 2nd edition, 1997.

[18] Donald W. Loveland. A variant of the Kolmogorov concept of complexity. *Information and Control*, 15:510–526, 1969.

[19] Per Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.

[20] J.S. Miller and L. Yu. On initial segment complexity and degrees of randomness. *Transactions-American Mathematical Society*, 360(6):3193, 2008.

[21] Keng Meng Ng. On strongly jump traceable reals. *Annals of Pure and Applied Logic*, 154:51–69, 2008.

[22] Keng Meng Ng. *Computability, Traceability and Beyond*. PhD thesis, Victoria University of Wellington, New Zealand, 2009.

[23] A. Nies. Eliminating concepts. *Computational prospects of infinity: Presented talks*, 2:225, 2008.

[24] A. Nies. *Computability and randomness*, volume 51. Oxford University Press, USA, 2009.

[25] André Nies. Reals which compute little. *Lecture Notes in Logic*, (27):261–275, 2002.

[26] André Nies. Lowness properties and randomness. *Advances in Mathematics*, 197:274–305, 2005.

[27] Piergiorgio Odifreddi. *Classical Recursion Theory, Volume 2. Number 143 in Studies in Logic and the Foundations of Mathematics*. North-Holland, 1999.

[28] H. Putnam. Trial and error predicates and the solution to a problem of Mostowski. *Journal of Symbolic logic*, pages 49–57, 1965.

[29] B. Russell. Les paradoxes de la logique. *Revue de Métaphysique et de Morale*, 14(5):627–650, 1906.

[30] Claus-Peter Schnorr. Process complexity and effective random tests. *Journal of Computer Systems Science*, 7:376–388, 1973.

[31] J. Shoenfield. On degrees of unsolvability. *Ann. of Math.*, 69:644–653, 1959.

[32] Robert I. Soare. *Recursively enumerable sets and degrees*. Springer, Heidelberg, 1987.

[33] Ray J. Solomonoff. A formal theory of inductive inference, Part I and Part II. *Information and Control*, 7:1–22 and 224–254, 1964.

[34] Robert Solovay. Draft of a paper (or series of papers) on Chaitin's work done for the most part during the period Sept. to Dec. 1974. Unpublished manuscript, IBM Thomas J. Watson Research Center, Yorktown Heights, New York. 215 pp., May 1975.

[35] Sebastiaan Terwijn and Domenico Zambella. Algorithmic randomness and lowness. *The Journal of Symbolic Logic*, 66:1199–1205, 2001.

[36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 42:230–265, 1936.

[37] Alan M. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, 2(1):161, 1939.

[38] Jean Ville. *Étude critique de la concept du collectif*. Gauthier-Villars, 1939.

[39] Richard von Mises. Grundlagen der Wahrscheinlichkeitsrechnung. *Mathematische Zeitschrift*, 5:52–99, 1919.