El método infalible para ganar a la ruleta en 21 días. ¡Llame YA!

Santiago Figueira

Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Charla de Borrachos 2005

Primera parte

Computabilidad

Historia de la Computabilidad

David Hilbert, 1900: Acá les dejo 23 problemas. ¿Es la matemática completa? Es decir, ¿puede ser demostrada o refutada cualquier sentencia matemática? y ¿es la matemática consistente? Además, ¿son las matemáticas decidibles?

Historia de la Computabilidad

David Hilbert, 1900: Acá les dejo 23 problemas. ¿Es la matemática completa? Es decir, ¿puede ser demostrada o refutada cualquier sentencia matemática? y ¿es la matemática consistente? Además, ¿son las matemáticas decidibles?

Kurt Gödel, 1931: Cualquier sistema formal suficientemente potente es inconsistente o incompleto. Además, si un sistema de axiomas es consistente, esta consistencia no puede demostrarse dentro del sistema formal.

Historia de la Computabilidad

David Hilbert, 1900: Acá les dejo 23 problemas. ¿Es la matemática completa? Es decir, ¿puede ser demostrada o refutada cualquier sentencia matemática? y ¿es la matemática consistente? Además, ¿son las matemáticas decidibles?

Kurt Gödel, 1931: Cualquier sistema formal suficientemente potente es inconsistente o incompleto. Además, si un sistema de axiomas es consistente, esta consistencia no puede demostrarse dentro del sistema formal.

David Hilbert: Chapeau!

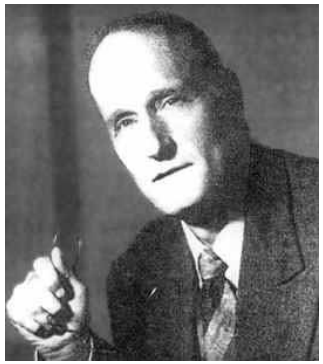
Alan Turing, 1936: Hola, yo soy Alan. Construí un modelo formal de cómputo, la Máquina de Turing, y demostré que hay problemas que esta máquina no puede resolver.

Alan Turing, 1936: Hola, yo soy Alan. Construí un modelo formal de cómputo, la Máquina de Turing, y demostré que hay problemas que esta máquina no puede resolver.

Alonzo Church, 1936: Yo hice lo mismo pero por un camino totalmente distinto. Me basé en una notación formal, que denominé cálculo lambda.

Alan Turing, 1936: Hola, yo soy Alan. Construí un modelo formal de cómputo, la Máquina de Turing, y demostré que hay problemas que esta máquina no puede resolver.

Alonzo Church, 1936: Yo hice lo mismo pero por un camino totalmente distinto. Me basé en una notación formal, que denominé cálculo lambda.



Stephen Kleene: Formulé la noción de función computable de otra manera. Estudié los grados de computabilidad de los conjuntos, la jerarquía aritmética, etc.

Emil Post: Trabajé en conjuntos recursivamente enumerables, reducibilidades y grados de computabilidad, etc.

Primer intento para *método efectivo*: funciones primitivas recursivas

Las funciones primitivas recursivas sólo pueden usar:

- cero
- sucesor $S(x) = x + 1$
- composición
- un tipo simple de recursión: si g y h son funciones primitivas recursivas, entonces también lo es f:

$$\begin{aligned} f(x, 0) &= g(x) \\ f(x, S(y)) &= h(x, y, f(x, y)) \end{aligned}$$

Primitivas recursivas = Programas con `for`

La función $+$ es primitiva recursiva porque

$$\begin{aligned} x + 0 &= x \\ x + S(y) &= S(x + y) \end{aligned}$$

Son funciones primitivas recursivas: el producto, la exponenciación las sumatorias, y predicados como

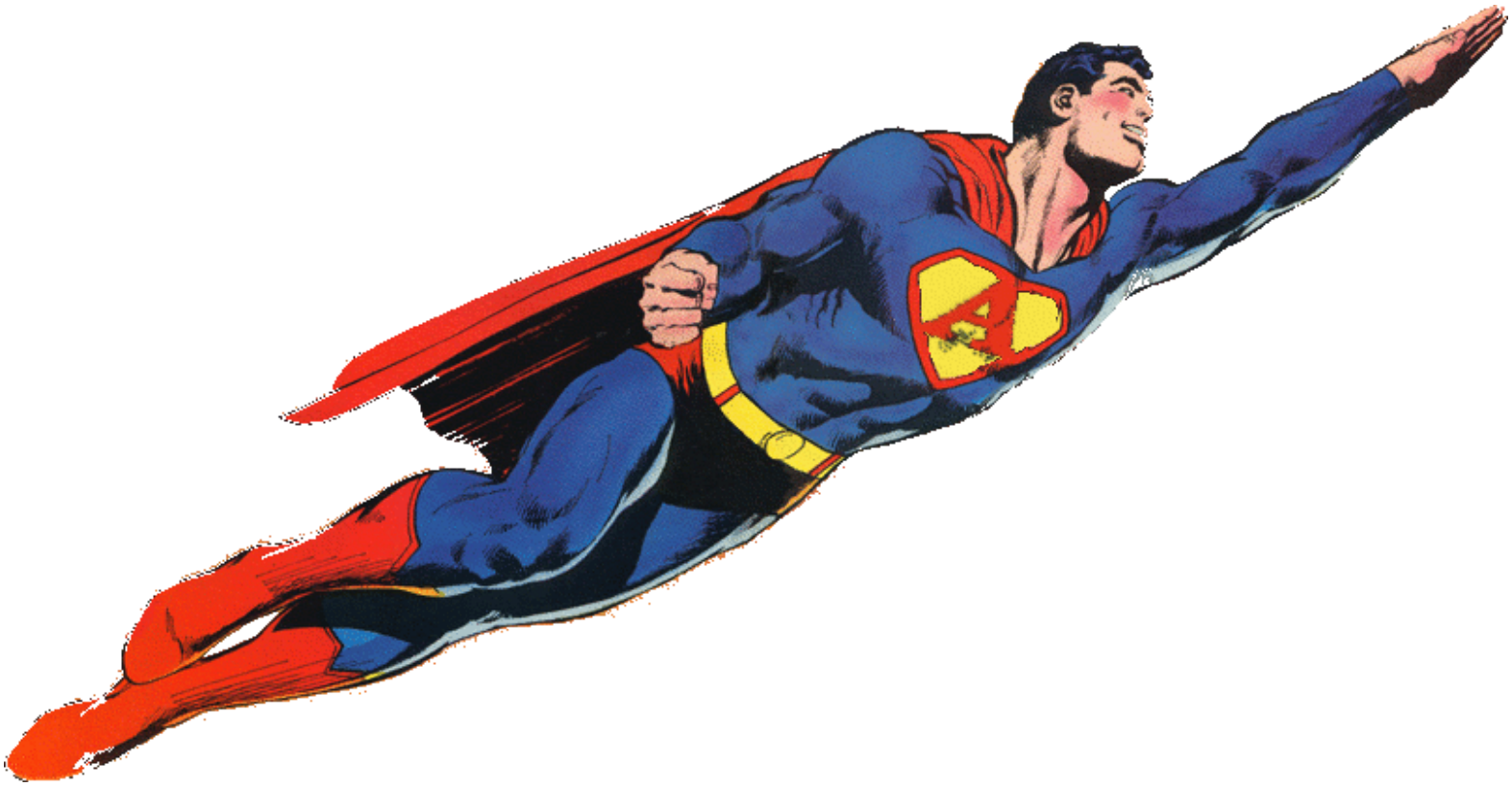
$$\text{esPrimo}(x) = \begin{cases} 1 & \text{si } x \text{ es primo} \\ 0 & \text{sino} \end{cases}$$

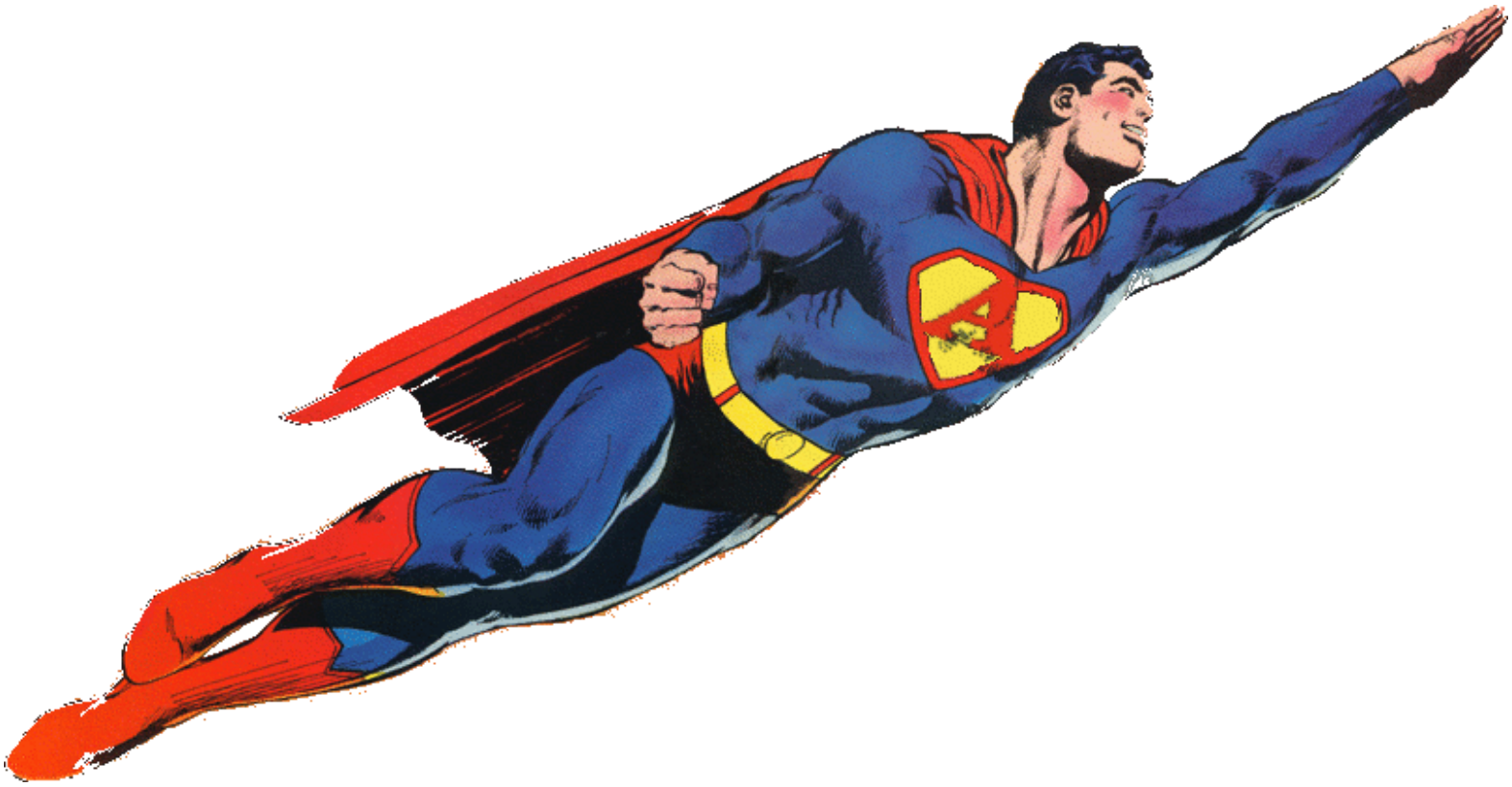
Las funciones primitivas recursivas coinciden con los programas que no usan `while` pero sí pueden usar `for`:

```
función factorial(n)
   r := 1
   for i:=1 to n
      r := r*i
   return r
```

¿Con esto basta?

¿Se puede definir cualquier función computable (total) usando programas `for`? O sea: ¿cualquier función (total) que programemos en una computadora es una función primitiva recursiva?


ACKERMANN



ACKERMANN

La función de ACKERMANN

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ y } n = 0 \\ A(m-1,A(m,n-1)) & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

La función de ACKERMANN

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ y } n = 0 \\ A(m-1, A(m,n-1)) & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

Para valores chicos de m, como 1, 2, o 3, A crece relativamente lento con respecto a n (a lo sumo exponencialmente).
Para $m \geq 4$, crece más rápido:

- $A(4,2) \simeq 2 \times 10^{19728}$
- $A(4,3) >$ cantidad de partículas del universo

La función de ACKERMANN

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ y } n = 0 \\ A(m-1, A(m,n-1)) & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

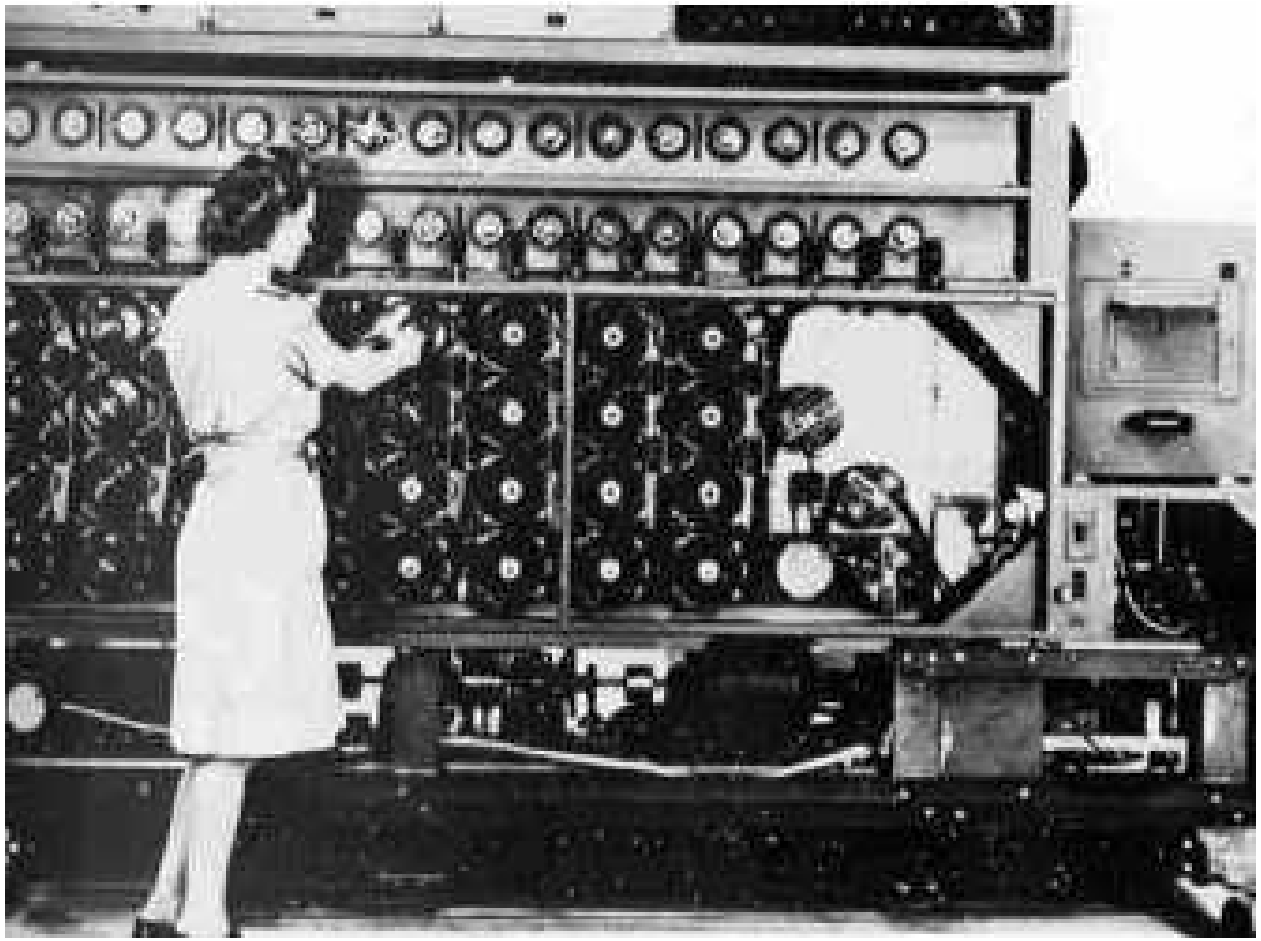
Para valores chicos de m, como 1, 2, o 3, A crece relativamente lento con respecto a n (a lo sumo exponencialmente).
Para $m \geq 4$, crece más rápido:

- $A(4,2) \simeq 2 \times 10^{19728}$
- $A(4,3) >$ cantidad de partículas del universo

$h(n) = A(n,n)$ crece más que la función exponencial, el factorial...

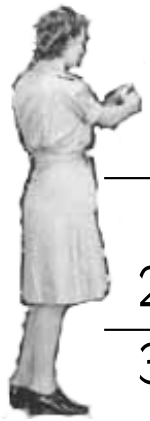
¡h crece más que cualquier función primitiva recursiva!

Segundo intento para *método efectivo*: máquinas de Turing



Las máquinas de Turing

Supongamos que la chica del slide anterior quiere calcular:



$$\begin{array}{cccc} & 2 & 3 & 1 \\ & \times & 1 & 3 \\ \hline & 6 & 9 & 3 \\ 2 & 3 & 1 & 0 \\ \hline 3 & 0 & 0 & 3 \end{array}$$

No es esencial:

- ¿mientras hace la cuenta está tomando un café?
- ¿escribe con lápiz o lapicera?
- ¿importa el tamaño del papel?

Lo importante es que:

- hace marcas en un papel
- para saber qué escribir en cada paso, le presta atención a lo que escribió antes

Las máquinas de Turing

Sin perder nada esencial, podemos suponer:

2	3	1	×	1	3	=	6	9	3	+	2	3	1	0	=	3	0	0	3

Empezamos con la multiplicación

2	3	1	×	1	3	=								...
		▲			▲									

Después de un tiempo, vamos a estar sumando

2	3	1	×	1	3	=	6	9	3	+	2	3	1	0	=		...
									▲					▲			

- En cada paso de la computación sólo un pequeño número de símbolos reciben atención (en el ejemplo, 2)
- La acción que se toma en cada paso depende sólo de los símbolos que están recibiendo atención y del estado actual de quien está haciendo el cálculo (en el ejemplo, la señorita está en estado *multiplicando* o *sumando*)

Entonces... ¿qué es un método efectivo?

Resultado básico de Teoría de la Computabilidad:
Son equivalentes:

- f es programable en algún lenguaje de programación (Java, C, etc.)
- f es computable por una máquina de Turing
- f es recursiva parcial (como primitivas recursivas pero con minimización, no lo vimos)
- f es definible en λ-cálculo (esto no lo vimos)

Conjuntos computables y recursivamente enumerables

Un conjunto $A \subseteq \mathbb{N}$ es computable cuando hay una función computable f tal que

$$f(n) = \begin{cases} 1 & n \in A \\ 0 & n \notin A \end{cases}$$

Un conjunto $A \subseteq \mathbb{N}$ es recursivamente enumerable (r.e.) cuando hay una función computable f tal que

$$A = \{f(n) \colon n \in \mathbb{N}\}$$

Conjuntos computables y recursivamente enumerables

Un conjunto $A \subseteq \mathbb{N}$ es computable cuando hay una función computable f tal que

$$f(n) = \begin{cases} 1 & n \in A \\ 0 & n \notin A \end{cases}$$

Un conjunto $A \subseteq \mathbb{N}$ es recursivamente enumerable (r.e.) cuando hay una función computable f tal que

$$A = \{f(n) \colon n \in \mathbb{N}\}$$

Cada programa tiene un número que lo identifica.

$$\mathcal{K} = \{e \colon \text{el } e\text{-ésimo programa evaluado en } e \text{ termina}\}$$

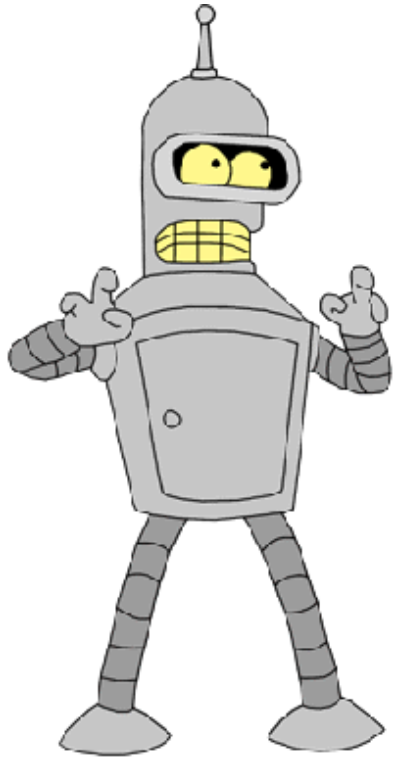
Es un conjunto r.e. pero no es computable.
Este es el famoso **halting problem**: *el problema de saber si un programa arbitrario termina no es computable*.

¿Qué hay además de las funciones computables?

Funciones primitivas recursivas

¿Qué hay además de las funciones computables?

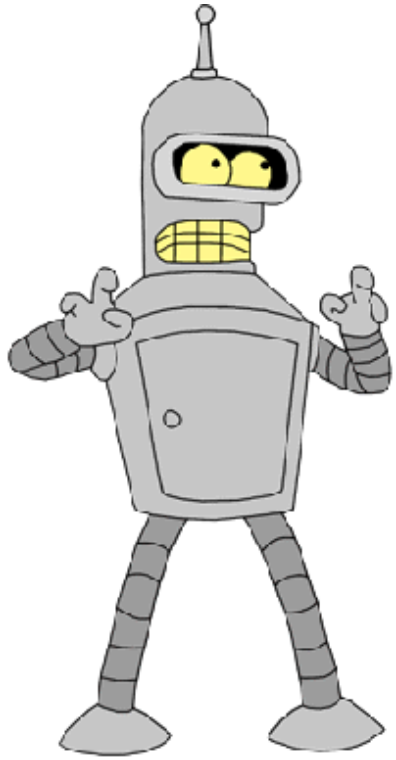
Funciones primitivas recursivas



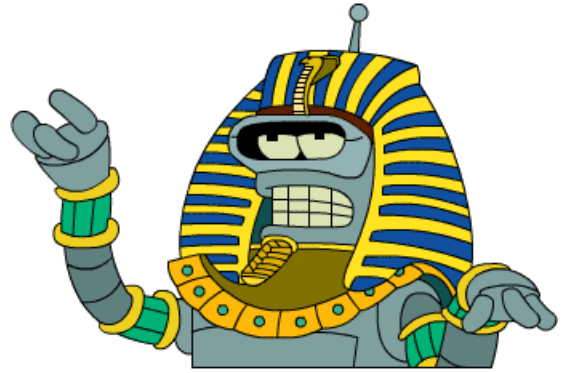
Funciones computables

¿Qué hay además de las funciones computables?

Funciones primitivas recursivas



Funciones computables



Cómputos con oráculo

Cómputos con oráculo

Son programas con una pequeña "ayudita".
Un programa con el oráculo para $B \subseteq \mathbb{N}$ tiene una instrucción nueva:

$$\text{estásEnB?}(n) = \begin{cases} 1 & n \in B \\ 0 & n \notin B \end{cases}$$

Un programa con oráculo para $\mathcal{K}$ es capaz de resolver el halting problem. Pero hay otros problemas que no se pueden resolver con la ayuda del oráculo $\mathcal{K}$.

Decimos $B \leq_T C$ cuando podemos computar B con un oráculo para C.
$\leq_T$ se llama reducibilidad de Turing.
Problema de Post: ¿Hay un conjunto X r.e. tal que $\emptyset <_T X <_T \mathcal{K}$?

Segunda parte

Aleatoriedad

(despiértense)

Aleatoriedad

Experimento: tirar una moneda 40 veces. Anotar

$$1 \longrightarrow \text{cara}$$
$$0 \longrightarrow \text{ceca}$$

¿Cuáles de estas secuencias parecen la salida de este experimento?

$$A = 0000000000000000000000000000000000000000$$
$$B = 1010101010101010101010101010101010101010$$
$$C = 1100010100110011010001101011101000101101$$

Aleatoriedad

Experimento: tirar una moneda 40 veces. Anotar

$$1 \longrightarrow \text{cara}$$
$$0 \longrightarrow \text{ceca}$$

¿Cuáles de estas secuencias parecen la salida de este experimento?

$$A = 0000000000000000000000000000000000000000$$
$$B = 1010101010101010101010101010101010101010$$
$$C = 1100010100110011010001101011101000101101$$

$$P(A) = P(B) = P(C) = 2^{-40}$$

¿Cómo se puede definir aleatoriedad?

Aleatoriedad = igual probabilidad para el 0 que para el 1.
Eliminamos la posibilidad

$$A = 00000000000000000000000000000000000000000$$

pero la secuencia

$$B = 10101010101010101010101010101010101010101$$

sería aleatoria. Demasiado débil.

Aleatoriedad = ausencia de patrones

La mayoría de las secuencias son aleatorias. Hay pocas secuencias con patrones reconocibles. Por eso, una secuencia como

$$B = 10101010101010101010101010101010101010$$

nos parece extraordinaria, pero una secuencia como

$$C = 11000101001100110100011010111010001011 01$$

no nos llama tanto la atención.

Posibles ataques a la noción de *ausencia de patrones*. Hay muchos, nosotros vamos a ver dos:

1. relacionado con juegos de azar, como la ruleta
2. relacionado con la compresión de datos

Martingalas

Supongamos una ruleta (sin el cero). Anotamos

$$\begin{aligned} 0 &\longrightarrow \text{rojo} \\ 1 &\longrightarrow \text{negro} \end{aligned}$$

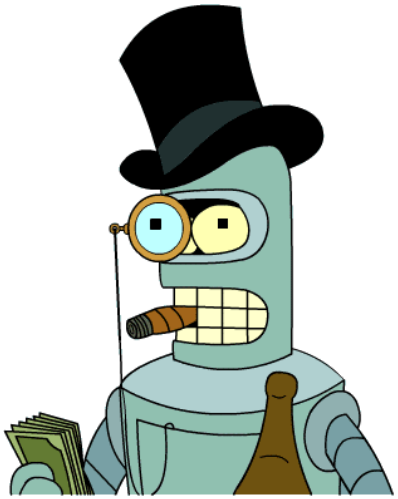
Idea: una secuencia (infinita) S **no es aleatoria** cuando hay formas de ganar en una ruleta en la que sale la secuencia de colores S.

Es fácil ganar en una ruleta que va sacando los colores:

$$\begin{aligned} A &= 000000000000000000000000000000000000000\ldots \\ B &= 101010101010101010101010101010101010101\ldots \end{aligned}$$

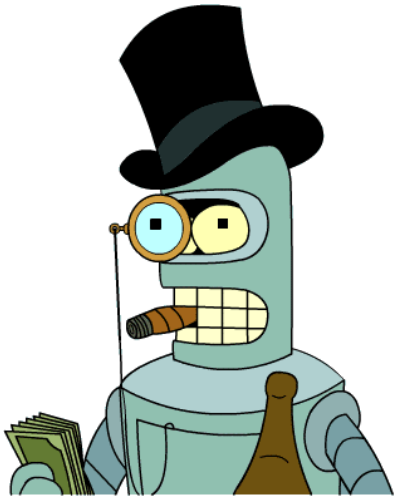
Esas secuencias no son aleatorias porque hay estrategias **computables** para apostar y ganar tanta plata como uno quiera.

Aleatorio = inexistencia de martingalas

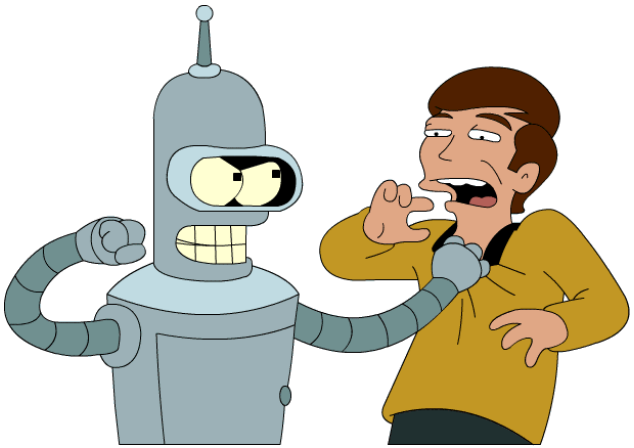


Una secuencia S **no es aleatoria** cuando existe una **martingala computable** que me permita ganar tanta plata como quiera, empezando con una cierta cantidad fija de plata y no pidiendo prestado.

Aleatorio = inexistencia de martingalas



Una secuencia S **no es aleatoria** cuando existe una **martingala computable** que me permita ganar tanta plata como quiera, empezando con una cierta cantidad fija de plata y no pidiendo prestado.



Una secuencia S **es aleatoria** cuando no existe ninguna **martingala computable** de esas características.

Paradoja de Berry

Los números pueden ser descriptos con oraciones:

- 235 = "doscientos treinta y cinco"
- 1048576 = "un millón cuarenta y ocho mil quinientos setenta y seis"; "dos a la veinte"
- 3628800 = "tres millones seiscientos veintiocho mil ochocientos"; "factorial de diez"

Como en castellano hay una cantidad finita de palabras, tiene que haber una cantidad finita de oraciones de menos de 15 palabras. Algunas de estas describen números y otras no.

> *el menor número que no puede ser descripto con menos de quince palabras.*

Por definición, no hay ninguna oración de menos de 15 palabras que lo describa.
¡Pero la oración de arriba lo describe y tiene 13 palabras!

Los programas describen palabras

Kolmogorov: los programas pueden ser vistos como descripciones algorítmicas de palabras.
Puedo describir

10101010101010101010101010101010101010101010101010101010101010

con los programas

```
return "101010101010
10101010101010101010
10101010101010101010
10101010"
```

```
r := ""
for i:=1 to 30
   r := r ++ "10"
return r
```

El primero tiene 69 caracteres, el segundo tiene 43.

Incompresibilidad

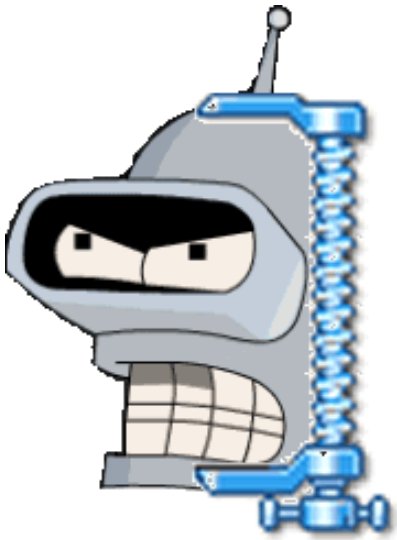
Idea: las secuencias aleatorias son difíciles de comprimir.
La secuencia infinita S es aleatoria cuando no hay programas cortos que describan a los prefijos de S.

$$
\begin{aligned}
A &= 000000000000000000000000000000000000000000\ldots \\
B &= 101010101010101010101010101010101010101010\ldots \\
C &= 110001010011001101000110101110100010110 1\ldots
\end{aligned}
$$

Es fácil comprimir segmentos iniciales de A y B. Lo único que se me ocurre para describir los primeros 10 y 20 bits de C es codificar la información textual adentro del programa.

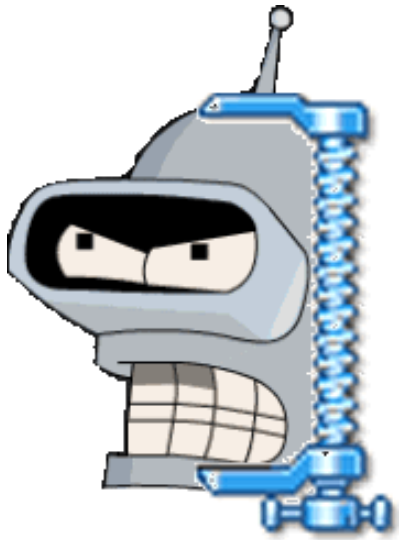
```
return "1100010100"        return "11000101001100110100"
```

Aleatorio = incompre(n)sible

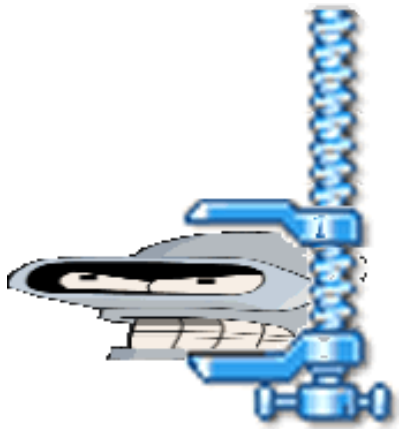


Una secuencia S **es aleatoria** cuando todos los prefijos de S son incompresibles.

Aleatorio = incompre(n)sible



Una secuencia S **es aleatoria** cuando todos los prefijos de S son incompresibles.



Una secuencia S **no es aleatoria** cuando puedo comprimir mucho los prefijos de S (por medio de programas).

Ejemplo de secuencia aleatoria

Una secuencia de 0s y 1s se puede ver como un número real en $[0, 1]$ escrito en binario. Chaitin probó que

$$\Omega = \sum_{\text{p no se cuelga}} 2^{-|p|} \in [0, 1]$$

es un real aleatorio.

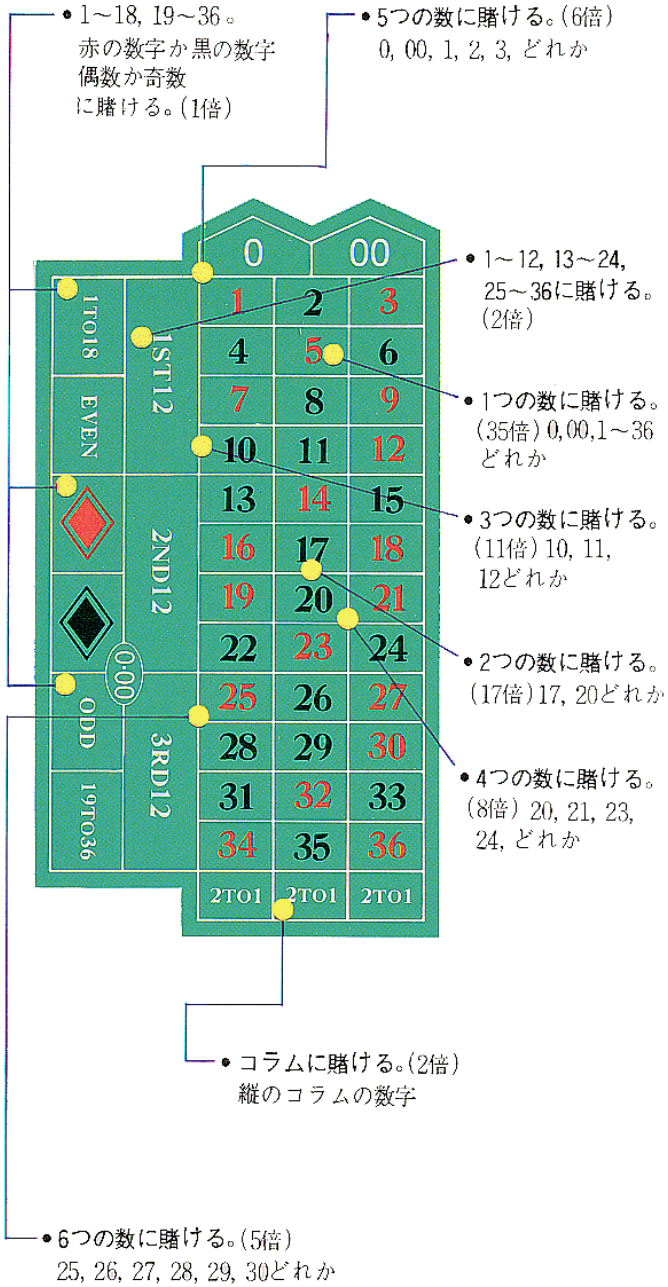
- no hay forma de ganar con una martingala computable en una ruleta que va tirando los bits de Ω
- no hay programas que describen de manera económica los prefijos de Ω

Además

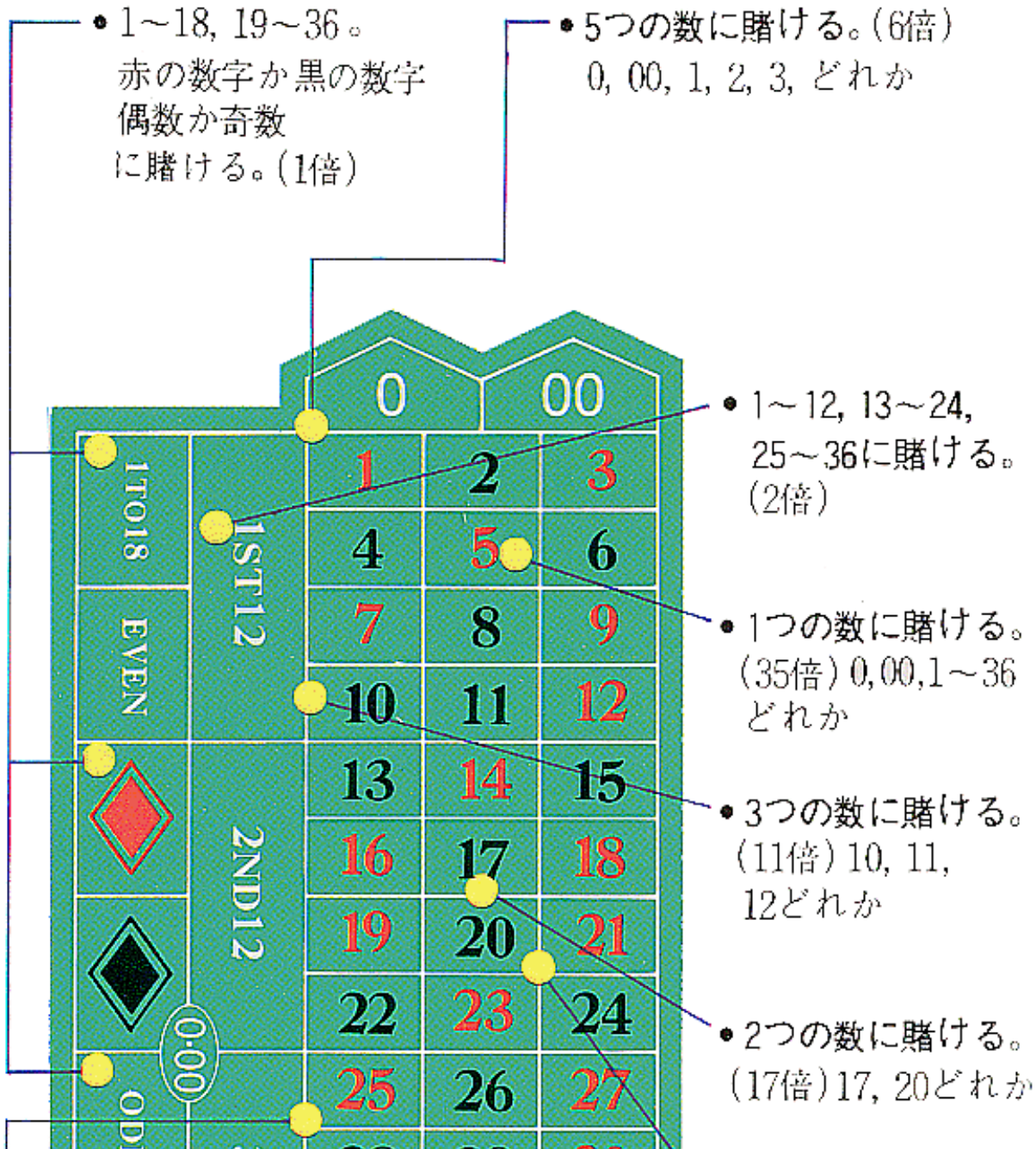
$$\Omega \leq_T \mathcal{K} \qquad \mathcal{K} \leq_T \Omega.$$

Entonces, Ω codifica al halting problem de manera muy compacta.

El método infalible para ganar a la ruleta – Explicación



El método infalible para ganar a la ruleta – Explicación



El método infalible para ganar a la ruleta

1. Apostar \$1 al rojo.
2. Si gané, volver al paso 1. Sino, apostar \$2 al rojo.
3. Si gané, volver al paso 1. Sino, apostar \$4 al rojo.
4. Si gané, volver al paso 1. Sino, apostar \$8 al rojo.
5. ...

Si sale negro, negro, negro, negro, ya perdí

$$\$1 + \$2 + \$4 + \$8 = \$15$$

Apuesto \$16 al rojo. Si sale rojo, gano \$16. Como había perdido \$15, mi ganancia es de \$1.
Entonces, cada vez que sale rojo tengo \$1 más.

El método infalible para ganar a la ruleta

1. Apostar \$1 al rojo.
2. Si gané, volver al paso 1. Sino, apostar \$2 al rojo.
3. Si gané, volver al paso 1. Sino, apostar \$4 al rojo.
4. Si gané, volver al paso 1. Sino, apostar \$8 al rojo.
5. ...

Si sale negro, negro, negro, negro, ya perdí

$$\$1 + \$2 + \$4 + \$8 = \$15$$

Apuesto \$16 al rojo. Si sale rojo, gano \$16. Como había perdido \$15, mi ganancia es de \$1.
Entonces, cada vez que sale rojo tengo \$1 más.
¡Les juro que esto funciona! (mi hermano lo hizo y con la plata que ganó se compró una bicicleta)
– ¿Por qué no es una martingala de las que vimos antes?

El método infalible para ganar a la ruleta

1. Apostar \$1 al rojo.
2. Si gané, volver al paso 1. Sino, apostar \$2 al rojo.
3. Si gané, volver al paso 1. Sino, apostar \$4 al rojo.
4. Si gané, volver al paso 1. Sino, apostar \$8 al rojo.
5. ...

Si sale negro, negro, negro, negro, ya perdí

$$\$1 + \$2 + \$4 + \$8 = \$15$$

Apuesto \$16 al rojo. Si sale rojo, gano \$16. Como había perdido \$15, mi ganancia es de \$1.
Entonces, cada vez que sale rojo tengo \$1 más.
¡Les juro que esto funciona! (mi hermano lo hizo y con la plata que ganó se compró una bicicleta)
– ¿Por qué no es una martingala de las que vimos antes?
– Porque no hay límite en la cantidad de plata que puedo perder en el medio. No hay garantías de que no me quede seco en la mitad del juego (mi hermano tuvo suerte...).