“Algunas demostraciones de incomputabilidad usando la técnica de diagonalziación”

Jesús Ibáñez, Arantza Irastorza, Ana Sánchez

UPV / EHU / LSI / TR 08-2000

Indice

Técnica de diagonalización

La disciplina informática ha evolucionado y evoluciona muy rápidamente. Sin embargo, sus fundamentos matemáticos, el núcleo de la llamada Informática Teórica, quedaron fijados en la década de los 30. Así, cuando aparecieron los primeros ordenadores basados tecnológicamente en la electrónica digital (a finales de los años 40) ya se habían establecido los supuestos fundamentales de la Teoría de la Computabilidad, supuestos que muchos años de vertiginoso cambio no han conseguido alterar. Alan Mathison Turing demostró ya entonces que ningún ordenador, por muy potente que lo imaginemos, podría resolver algunas cuestiones. Estos problemas para los que no existe ningún algoritmo posible, *los incomputables,* no son excepcionales ni patológicos, son abundantes y palpables, y hay un gran número de ellos entre los problemas que se plantean en torno al comportamiento de los programas. Por ejemplo, no existe un algoritmo que, dados dos programas, nos diga si son equivalentes, es decir, si producen los mismos resultados cuando operan bajo las mismas condiciones iniciales. Tampoco lo hay para anticipar si la ejecución de un programa va a pasar por una instrucción concreta o no. El problema de parada, sin duda el miembro más conocido de esta familia, es también el "responsable" primario de la incomputabilidad de los antes citados y de la de muchos otros: no existe un algoritmo para decidir con carácter general si un programa ciclará o no al recibir unos datos de entrada concretos. Es el primer ejemplo de problema incomputable y como tal, capítulo obligado en cualquier libro de Teoría de la Computabilidad [Har 87, MAK 88, SW 88].

A diferencia de lo que sucede cuando se pretende probar la computabilidad de un problema, para lo cuál es suficiente con encontrar un algoritmo concreto que lo resuelva, al enfrentarnos a la tarea de demostrar su incomputabilidad necesitamos un argumento lógico que certifique la inexistencia de tal algoritmo, o lo que es lo mismo, que pruebe que ninguno de los algoritmos existentes es capaz de resolver dicho problema. Tal argumento de carácter universal no suele ser sencillo de establecer, y normalmente suele estar relacionado con una demostración por reducción al absurdo. Existen distintas técnicas para lograr este objetivo, siendo la de *diagonalización*, la de *reducción* y la del *punto fijo* las más importantes.

La técnica de diagonalización es la más básica de ellas, y resulta bastante conocida al no tratarse de una herramienta específica de la Informática Teórica. Este tipo de demostración formal, que se aplicó por primera vez en 1873 en Teoría de Conjuntos para demostrar la existencia de conjuntos no numerables (de cardinal superior al del conjunto de los números naturales), se debe a Georg Ferdinand Ludwig Philipp Cantor. Su técnica combina la demostración por contradicción con un proceso

de construcción intermedio especial [Sta 81]. Es necesario disponer de una clase infinita y numerable de objetos $\{O_1, O_2, O_3, ...\}$, cada uno de los cuáles se puede definir mediante una lista infinita numerable de atributos independientes $O_i = (a_{i1}, a_{i2}, a_{i3}, ...)$. La contradicción se busca construyendo un nuevo objeto O' que debiera, por su naturaleza, pertenecer a la clase citada, pero que sin embargo se diferencia de cada elemento individual de la clase O_i precisamente en su *i*-ésimo atributo a_{ii}. El absurdo surge de las propiedades contradictorias que el *objeto diagonal* O' hereda como miembro de la clase y como no-miembro de la misma.

En el uso que aquí hacemos de la técnica de diagonalización los objetos serán las funciones computables, y los atributos que definen a una función serán los resultados que dicha función asocia a cada uno de los datos posibles. Nuestro objetivo consistirá en construir una *función computable diagonal* diferente de todas las existentes: la nueva función se distinguirá de la *i*-ésima función computable porque tendrá un resultado distinto para el *i*-ésimo dato de entre los posibles.

En nuestro caso la técnica también se relaciona con la autorreferencia a causa del fenómeno de la gödelización de los programas. El absurdo suele encontrarse cuando, tras comprobar que la función diagonal debe estar en la clase de las computables y por tanto le corresponde un cierto lugar *e* en la lista de las funciones, tratamos de averiguar su comportamiento sobre el *e*-ésimo dato. Pero como sucede que las funciones se ordenan según los programas que las computan, la *e*-ésima función es la computada por el programa que ocupa el lugar *e*. Por otro lado, los programas han sido ordenados utilizando un esquema de representación que permite asociarlos biunívocamente con los datos, por lo que el *e*-ésimo programa es representado por el *e*-ésimo dato. La conclusión es que, en definitiva, el absurdo surge al aplicar el programa de lugar *e* sobre sí mismo. Este esquema autorreferencial es el responsable del paralelismo entre muchas formulaciones diagonales y las paradojas Russellianas.

De todas formas, en este documento no tratamos de explicar la técnica en sí, que se supone conocida, sino de ilustrarla con una colección de ejemplos de diferente grado de dificultad. Sabemos que la percepción sobre la dificultad es subjetiva, pero hemos tratado de ordenarlos de forma que el aprendizaje sea incremental. Aún así cada ejemplo es autocontenido, por lo que podrían consultarse en cualquier orden.

Nociones básicas y notación

Para describir el comportamiento de los programas en términos de entrada/salida hemos de utilizar funciones parciales, que pueden estar indefinidas para algunas entradas. Si la función ψ aplicada sobre x converge (tiene imagen) lo indicamos como $\psi(x)\downarrow$. Si, por el contrario, la función está indefinida en ese punto lo denotamos como $\psi(x)\uparrow$. En general las funciones pueden tener cualquier número de argumentos, pero para aplicar la técnica de diagonalización conviene reducirlas a funciones de una única entrada para simplificar la construcción de la función diagonal. Es por ello que todos los ejemplos que se incluyen en este documento son de este tipo.

Una función es *computable* si existe un programa en algún lenguaje de programación que la calcula. El lenguaje empleado para demostrar la computabilidad en este documento es el de los programas-while, sobre el que se ha definido un lenguaje de macros que permite abreviar y entender mejor el significado de los programas [IIS 96]. No obstante, es sabido que cualquier sistema de programación empleado para demostrar la computabilidad define el mismo conjunto de funciones computables. En el lenguaje de los programas-while los datos sobre los que se opera son las palabras o cadenas de caracteres sobre un alfabeto arbitrario $\sum$. Sin embargo, es posible definir sencillas correspondencias entre otros conjuntos posibles de datos y los elementos de $\sum^*$, de forma que cada cadena represente un elemento del nuevo conjunto y que la computación sobre palabras sea congruente con la representación de los elementos del nuevo tipo. Algunos de los tipos de datos así implementados son los booleanos $\mathbb{h}$, los números naturales $\mathbb{V}$ y los propios programas-while '. Salvo en el primer caso por razones obvias, las implementaciones se hacen siempre biyectivas.

Una vez implementados los números naturales podemos enumerar el conjunto de posibles datos como $\sum^* = \{w_0, w_1, w_2, w_3, ...\}$, donde w_i es la palabra que sirve para representar el número *i*. A su vez, podemos enumerar el conjunto de los programas-while como ' = $\{P_0, P_1, P_2, P_3, ...\}$, donde P_i es el programa representado por la palabra w_i. Se dice entonces de w_i (y, por extensión, también de *i*) que es el *código* del programa P_i.

Dado que en esta enumeración están todos los programas posibles, también podemos enumerar en una lista la clase de todas las funciones computables de un argumento $\{\varphi_0, \varphi_1, \varphi_2, \varphi_3, ...\}$, donde φ_i es la función computada por el programa P_i cuando se le suministra un único dato. Se dice entonces de w_i (y, por extensión, también de *i*) que es un *índice* de la función φ_i. Además nos referimos al dominio de la función φ_i como W_i y a su rango como R_i.

Por todo ello, afirmar que una función ψ es computable equivale a aseverar que existe un índice *e* tal que $\psi \cong \varphi_e$, es decir, a indicar que debe tener un lugar en la enumeración citada más arriba.

En el conjunto de las funciones computables hay una que requiere especial atención porque se usará frecuentemente en este documento: *la función universal*. Esta función es un intérprete de programas-while, y por tanto es capaz de simular el comportamiento de cualquier función computable sobre cualquier entrada. La función universal se denota con la letra Φ y tiene la siguiente definición formal

$$\Phi(x, y) \cong \begin{cases} \varphi_x(y) & \varphi_x(y)\downarrow \\ \perp & \text{c.c.} \end{cases}$$

De la misma manera que podemos clasificar las funciones en computables o no computables, podemos distinguir dos clases entre los conjuntos: la clase de los conjuntos recursivos o decidibles, $\sum_0$, y la clase de los conjuntos recursivamente enumerables o semidecidibles, $\sum_1$. Un conjunto A cualquiera tiene asociadas dos funciones: su función característica C_A y su función semicaracterística χ_A. Ambas funciones se definen como sigue:

$$C_A(x) = \begin{cases} \text{true} & x \in A \\ \text{false} & x \notin A \end{cases}$$

$$\chi_A(x) \cong \begin{cases} \text{true} & x \in A \\ \perp & x \notin A \end{cases}$$

Para cada conjunto A, decimos que es *decidible* si su función característica es computable, y que es *semidecidible* si su función semicaracterística es computable.

Un resultado importante es que los elementos de un conjunto semidecidible y no vacío A pueden ser definidos por una función f computable y total, de forma que dicho conjunto sea igual al rango de f. Esta propiedad se puede utilizar para aplicar la técnica de diagonalización, ya que de está manera podremos disponer de los elementos de A mediante la enumeración A = ran(f) = {f(0), f(1), f(2), f(3), …}

La función $\psi(x) = \begin{cases} x+1 & \varphi_x(x)+1>3 \\ 0 & \text{c.c.} \end{cases}$ no es computable

Si suponemos que ψ es computable, entonces podemos saber, para cualquier programa de código **x**, si $\varphi_x(\mathbf{x})>2$, ya que bastará ver si $\psi(\mathbf{x})>0$. Basándonos en ello podremos definir una función diagonal ρ computable, pero que será diferente de todas las funciones computables. Para establecer quién será $\rho(\mathbf{x})$ para cada valor **x** tendremos en cuenta lo siguiente:

- Si $\psi(\mathbf{x})>0$ entonces sabemos que $\varphi_x(\mathbf{x})+1>3$. Por tanto, podemos hacer que ρ sea diferente de φ_x en el punto **x** definiendo $\rho(\mathbf{x})$ con un valor que no sea mayor que 2, por ejemplo 0 (podríamos haber elegido cualquier otro valor menor o igual a 2, o también dejar el valor indefinido).
- Si $\psi(\mathbf{x})=0$ entonces sabemos que $\neg(\varphi_x(\mathbf{x})+1>3)$, lo que significa que, o bien $\varphi_x(\mathbf{x})$ es menor o igual a 2, o bien diverge. Por tanto podemos hacer que ρ sea diferente de φ_x en el punto **x** definiendo $\rho(\mathbf{x})$ con un valor mayor que 2, por ejemplo 5.

		$\Psi(0)=0$	$\Psi(1)>0$	$\Psi(2)=0$	$\Psi(3)>0$	
x	ρ	φ_0	φ_1	φ_2	φ_3	
0	5	$\neg(\varphi_0(0)>2)$	?	?	?	
1	0	?	$\varphi_1(1)>2$	?	?	
2	5	?	?	$\neg(\varphi_2(2)>2)$	?	…
3	0	?	?	?	$\varphi_3(3)>2$	
		?	?	?	?	

…

El esquema se muestra en la tabla-ejemplo, donde vemos que los puntos en los que la función que construimos difiere de las otras funciones recorren la diagonal de la tabla. En la fila superior se indica resultados que nos podría proporcionar ψ al ser aplicada sobre cada índice **x**. En la zona sombreada de la tabla vemos qué información obtenemos sobre valores de la misma (coinciden con la diagonal). Por último, en la

columna izquierda se indican los valores que tomaría ρ para distinguirse de todas y cada una de las funciones de la lista.

Vamos a completar la demostración siguiendo el esquema que acabamos de explicar. Supongamos que ψ es computable. Entonces definimos la siguiente función:

$$\rho(x) = \begin{cases} 5 & \psi(x) = 0 \\ 0 & \psi(x) > 0 \end{cases}$$

que es computable, como muestra el programa dado a continuación:

if ψ(X1) = 0 **then** X0 := 5; **else** X0 := 0; **end if**;

El programa anterior, una vez expandidas sus macros y codificado el programa-while resultante, tendrá un cierto código **e**, por lo que $\rho = \varphi_{\mathbf{e}}$. Por la definición de la función sabemos que $\forall y(\rho(y)=0 \vee \rho(y)=5)$, pero si tratamos de comprobar cuál es el resultado de aplicar esta función sobre su propio índice llegamos a un absurdo:

$$\rho(\mathbf{e})=0 \overset{\rho=\varphi_e}{\Rightarrow} \varphi_{\mathbf{e}}(\mathbf{e})=0 \Rightarrow \neg(\varphi_e(\mathbf{e})>2) \overset{def.\Psi}{\Rightarrow} \Psi(\mathbf{e})=0 \overset{def.\rho}{\Rightarrow} \rho(\mathbf{e})=5 \Rightarrow \neg\rho(\mathbf{e})=0$$

$$\neg\rho(\mathbf{e})=0 \overset{def.\rho}{\Rightarrow} \rho(\mathbf{e})=5 \overset{\rho=\varphi_e}{\Rightarrow} \varphi_{\mathbf{e}}(\mathbf{e})=5 \Rightarrow \varphi_e(\mathbf{e})>2 \overset{def.\Psi}{\Rightarrow} \Psi(\mathbf{e})=\mathbf{e}+1>0 \overset{def.\rho}{\Rightarrow} \rho(\mathbf{e})=0$$

Hemos demostrado que ρ(**e**) no puede ni ser el valor 0 ni dejar de serlo, por lo que hemos llegado a un absurdo. Este absurdo proviene de la suposición que hemos hecho de que la función ψ es computable, con lo que queda demostrado que no puede serlo.

$A = \{ x: \forall y\ (y \leq x \rightarrow \varphi_x(y) \neq y) \}$ no es decidible

Suponemos que el conjunto A es decidible, y por tanto que podemos saber, dado un programa de código **x**, si éste pertenece o no al conjunto A, al ser su función característica C_A computable. Basándonos en esta hipótesis podemos definir, por diagonalización, una función ψ computable diferente de todas las funciones computables. Haremos que la función ψ y cada una de las funciones computables $\varphi_{\mathbf{x}}$ se diferencien por lo menos en un punto (si podemos, el de la diagonal **x**).

- Si **x** está en A (cosa que averiguamos porque $C_A(\mathbf{x}) = \text{true}$), sabremos que para todas las palabras **y** menores o igual a **x** la función $\varphi_{\mathbf{x}}$ converge y devuelve un valor diferente a **y**. Es decir que en particular para **x**, $\varphi_x(\mathbf{x}) \neq \mathbf{x}$. Entonces podríamos definir $\psi(\mathbf{x})$ de muchas maneras, por ejemplo, **x** ó $\perp$, y tendríamos con ello que $\neg\psi(x) \cong \varphi_x(\mathbf{x})$; también lo conseguiríamos definiéndolo como $\varphi_{\mathbf{x}}(\mathbf{x})+100$, $\varphi_{\mathbf{x}}(\mathbf{x})+1$, $\varphi_{\mathbf{x}}(\mathbf{x})*2+2$, ya que sabemos que $\varphi_{\mathbf{x}}(\mathbf{x})$ converge.

- Si **x** no está en A (cosa que averiguamos porque $C_A(\mathbf{x}) = \text{false}$), sabremos que para alguna de las palabras **y** menores o igual a **x** la función $\varphi_{\mathbf{x}}$ diverge o devuelve un valor igual a **y**. Pero no sabemos sobre qué palabra en concreto, ni tampoco lo que hace sobre las demás. Por tanto no tenemos información alguna sobre el comportamiento de $\varphi_{\mathbf{x}}$ sobre la palabra **x**, y no parece que podamos definir $\psi(\mathbf{x})$ de forma que sea diferente a $\varphi_{\mathbf{x}}$ en ese punto.

Aunque en este segundo caso no podemos conseguir que las funciones ψ y $\varphi_{\mathbf{x}}$ se diferencien localmente en el punto **x**, podemos intentar definir ψ de forma que se diferencie globalmente de $\varphi_{\mathbf{x}}$. Lo único que sabemos de $\varphi_{\mathbf{x}}$ en este caso es que su índice **x** no pertenece a A, por lo que una buena solución sería conseguir que todos los índices de ψ pertenezcan a A. De este modo sabremos seguro que $P_{\mathbf{x}}$ no es uno de los programas que computa a ψ, por lo que esta será diferente de $\varphi_{\mathbf{x}}$ (aunque no sabremos en qué punto concreto).

Haremos que nuestra función diagonal sea convergente y devuelva un valor diferente al de la entrada no únicamente para los valores menores o iguales que una cierta cota, sino para todos los datos. Así tendremos la seguridad de que todos sus índices, por grandes que sean, cumplirán las restricciones que impone el conjunto A. Dado que va a ser total, será más adecuado llamarla h.

- Si **x** está en A, definiremos $h(\mathbf{x})$ como $\varphi_{\mathbf{x}}(\mathbf{x})+\mathbf{x}+1$, con lo que será diferente de **x** además de serlo de $\varphi_{\mathbf{x}}(\mathbf{x})$. Podemos hacer esto porque sabemos que $\varphi_{\mathbf{x}}(\mathbf{x})\downarrow$

- Si **x** no está en A no necesitamos el punto **x** para diferenciarnos localmente de ninguna función computable. Sin embargo, definiremos h(**x**) como **x**+1 (o cualquier valor diferente de **x**) para conseguir que los índices de nuestra función h pertenezcan a A, y así se diferencie globalmente de φ_x.

El esquema quedaría ilustrado en la siguiente tabla-ejemplo, donde podemos ver que la nueva función se diferencia de algunas funciones en el punto de la diagonal y de otras globalmente: no podemos determinar en qué punto, pero lo seguro es que los índices de una están en A y los de la otra no. La definición formal de la función es como sigue:

$$h(x) \cong \begin{cases} \varphi_x(x) + x + 1 & x \in A \\ x + 1 & x \notin A \end{cases}$$

		$\notin A$	$\in A$	$\notin A$	$\in A$	$\notin A$	$\notin A$	$\in A$	
x	h	φ_0	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	…
0	**0+1**	?	≠0	?	≠0	?	?	≠0	
1	**φ_1(1)+1+1**	?	≠1	?	≠1	?	?	≠1	
2	**2+1**	?	?	?	≠2	?	?	≠2	
3	**φ_3(3)+3+1**	?	?	?	≠3	?	?	≠3	. . .
4	**4+1**	?	?	?	?	?	?	≠4	
5	**5+1**	?	?	?	?	?	?	≠5	
6	**φ_6(6)+6+1**	?	?	?	?	?	?	≠6	

. . .

Suponiendo que A es decidible, la función que hemos definido puede ser computada por el siguiente programa:

if C_A(X1) **then** X0 := Φ(X1, X1) + X1+1; **else** X0 := X1+1; **end if;**

Una vez expandidas sus macros este programa se convertirá en un programa while, cuyo código será un índice e de la función h, es decir, que $h \cong \varphi_{\mathbf{e}}$. Dado que para todas las entradas la función h converge y siempre devuelve un valor que es diferente al de la entrada, se tiene que $\mathbf{e} \in \mathbf{A}$, ya que $\forall y\ (h(y)\downarrow \wedge h(y)>y)$.

Veamos cuál es el comportamiento de la función sobre su propio índice **e**:

$$\varphi_e(\mathbf{e}) \overset{h \cong \varphi_e}{=} h(\mathbf{e}) \overset{\text{def. } h}{=} \varphi_{\mathbf{e}}(\mathbf{e})+\mathbf{e}+1$$

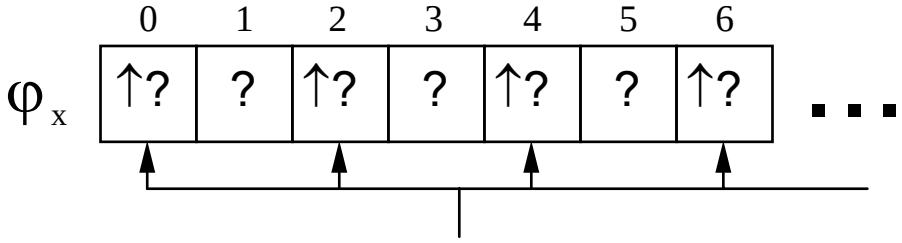
La única posibilidad de que $\varphi_e(\mathbf{e}) = \varphi_{\mathbf{e}}(\mathbf{e})+e+1$ es que $\varphi_e(e)\uparrow$, pero esto contradice el hecho de que $h \cong \varphi_e$ es total.

Todo el razonamiento que hemos seguido es correcto, luego la contradicción viene del hecho de suponer que A es decidible, por lo que queda demostrado que esto no es posible.

El conjunto { x: ∀y $\varphi_x(2*y)\downarrow$ } no es decidible

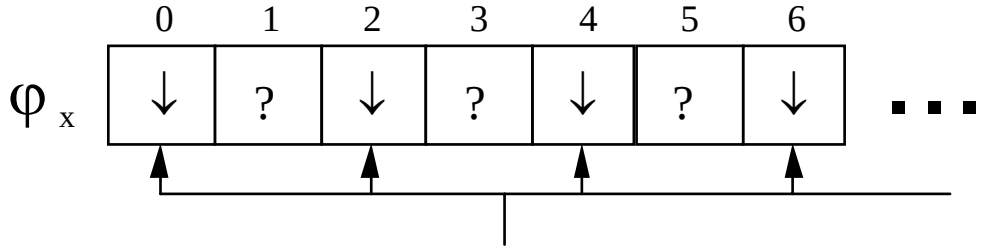
Llamemos A al conjunto dado, que contiene a todos los programas que convergen sobre todos los datos pares. Debemos suponer que A es decidible, y basándonos en ello construir una función diagonal **h** distinta de todas las funciones computables. Con esta suposición, dado un programa cualquiera de código **x**, podemos preguntar si está en A o no.

- Si **x** no está en A, la información que obtenemos sobre **x** es mínima: P_x diverge ante algún dato par, pero no sabemos ante cuál. Por tanto, tenemos muy difícil hacer que **h** sea diferente de φ_x en un dato concreto. Como lo único que sabemos de φ_x es que diverge para algún par, en lugar de intentar diferenciar **h** de φ_x justamente en el punto **x**, podemos intentar conseguir que **h** converja para todos los datos pares, diferenciándose así de φ_x de manera global.



Sabemos que φ_x diverge en alguna de estas posiciones,
pero no sabemos en cuál ni qué hace en las demás

- Si **x** está en A, sabemos que φ_x converge para todos los datos pares. Ello nos da una información muy valiosa para conseguir que **h** sea diferente de φ_x.



Sabemos que φ_x converge en todas estas posiciones,
por lo que podemos calcular los resultados que nos
interesen y obrar en consecuencia

Queda un pequeño problema: Si **x** ∈ A y además es par sabemos que $\varphi_x(\mathbf{x})\downarrow$, por lo que podemos calcular su valor y conseguir que **h**(**x**) sea diferente. Pero si **x** es impar no sabemos nada de $\varphi_x(\mathbf{x})$, y estamos igual de vendidos/as que antes. Una forma de

solucionar esto es hacer que **h** se diferencie de $\varphi_{\mathbf{x}}$ no en el punto **x**, sino en el punto $2*\mathbf{x}$, donde siempre tenemos convergencia.

En resumen, nuestra estrategia de construcción de la función diagonal será:

- Si **x** es impar no tenemos restricciones, ya que ese valor no es utilizado para diferenciar h de ninguna $\varphi_{\mathbf{x}}$ (elegimos devolver 0).
- Si **x** es par tenemos dos obligaciones. Por un lado, es necesaria la convergencia, para asegurar que **h** se diferencia globalmente de las funciones cuyos índices no están en A. Al mismo tiempo, el valor que **h** produzca en el valor par **x** quizá deba ser usado para diferenciarse localmente de la función $\varphi_{\mathbf{x/2}}$. Si $\mathbf{x}/2 \in A$, esta diferenciación local es necesaria y, dado que **x** es par, $\varphi_{\mathbf{x/2}}(\mathbf{x})\downarrow$, y entonces nuestra función **h** ha de devolver un valor diferente de este, por ejemplo $\varphi_{\mathbf{x/2}}(\mathbf{x})+1$. En cambio, si $\mathbf{x}/2 \notin A$ no nos preocupamos, ya que **h** se diferenciará de $\varphi_{\mathbf{x/2}}$ globalmente (elegimos también 0).

Es necesario incidir en la importancia de comparar **h** con la función de índice $\mathbf{x}/2$. Al utilizar los valores pares de **x** (0, 2, 4, 6, …), comparar **h** con $\varphi_{\mathbf{x/2}}$ significa poder comparar **h** con todas las funciones computables (φ_0, φ_1, φ_2, φ_3,…) y conseguir que **h** se pueda diferenciar de todas ellas. El esquema quedaría ilustrado por la tabla-ejemplo de la página siguiente.

Vamos a utilizar la construcción mencionada para completar la demostración. Supongamos que A es decidible. Entonces la siguiente función será computable:

$$h(x) = \begin{cases} 0 & \text{si } x \bmod 2 = 1 & \text{(podría ser cualquier cosa)} \\ 0 & \text{si } x \bmod 2 = 0 \wedge x/2 \notin A & \text{(basta con converger)} \\ \varphi_{x/2}(x) + 1 & \text{si } x \bmod 2 = 0 \wedge x/2 \in A & \text{(para diferenciar h localmente)} \end{cases}$$

como muestra el programa que figura a continuación

if X1 mod 2 = 0 **and** C_A(X1 / 2) **then** X0 := Φ(X1 / 2, X1) + 1; **else** X0 := 0; **end if;**

El programa anterior, una vez expandidas sus macros, tendrá un cierto código **e**. Como por su definición $\forall y\ \mathbf{h}(2*y)\downarrow$, entonces $\forall y\ \varphi_{\mathbf{e}}(2*y)\downarrow$. De todo ello concluimos al menos tres cosas: $\mathbf{h} = \varphi_{\mathbf{e}}$, $\mathbf{e} \in A$ y $\varphi_{\mathbf{e}}(2*\mathbf{e})\downarrow$. Pero si tratamos de calcular este último valor:

$$\left.\begin{array}{l} 2*\mathbf{e} \bmod 2 = 0 \\ 2*\mathbf{e}/\ 2 = \mathbf{e} \in A \end{array}\right\} \Rightarrow \varphi_{\mathbf{e}}(2*\mathbf{e}) \overset{\varphi_e = h}{=} h(2*\mathbf{e}) \overset{\text{def. h}}{=} \varphi_{2*\mathbf{e}/2}(2*\mathbf{e}) + 1 = \varphi_{\mathbf{e}}(2*\mathbf{e})+1 \Rightarrow \varphi_{\mathbf{e}}(2*\mathbf{e})\uparrow$$

Por un lado tenemos que $\varphi_{\mathbf{e}}(2*\mathbf{e})\downarrow$, y por otro que $\varphi_{\mathbf{e}}(2*\mathbf{e})\uparrow$. Esta contradicción viene del hecho de suponer que A es decidible, por lo que queda demostrado que no es posible.

		$\in A$	$\in A$	$\notin A$	$\in A$	$\notin A$	$\in A$	$\notin A$
x	**h**	φ_0	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6
0	1	0	3	?	0	?	7	?
1	0	?	?	?	?	?	?	?
2	5	0	4	?	2	?	7	?
3	0	?	?	?	?	?	?	?
4	0	0	2	?	4	?	7	?
5	0	?	?	?	?	?	?	?
6	7	0	7	?	6	?	4	?
7	0	?	?	?	?	?	?	?
8	0	0	8	?	8	?	4	?
9	0	?	?	?	?	?	?	?
10	4	0	4	?	10	?	3	?

$A= \{ x: W_X=\{y: y \bmod 2 = 0\} \}$ no es decidible

Suponemos que el conjunto A es decidible, es decir, que su función característica C_A es computable. De esta manera podremos preguntar, dada una función de índice **x**, si éste pertenece o no al conjunto A, y basándonos en ello definir una función diagonal Ψ computable. Para construirla haremos que Ψ sea diferente de todas las funciones computables asegurándonos de que las funciones Ψ y φ_x se diferencien por lo menos en un punto.

- Si **x** está en A sabremos que el dominio de la función $\varphi_{\mathbf{x}}$ es el conjunto de todos los pares (es decir, que si **x** es par $\varphi_{\mathbf{x}}(\mathbf{x})\downarrow$, y que si es impar $\varphi_{\mathbf{x}}(\mathbf{x})\uparrow$), y podremos definir $\Psi(\mathbf{x})$ de forma que, por ejemplo, en el primer caso diverja y en el segundo converja.

- Si $\mathbf{x} \notin$ A sólo sabremos que la función tiene un dominio que no coincide con el de los valores pares, pero no podemos saber si ello es porque $\varphi_{\mathbf{x}}$ diverge para algún par o porque converge para algún impar. Tendremos información acerca del comportamiento global de la función, pero no habrá manera de saber qué hace sobre la palabra **x** ni sobre ninguna otra (en particular no sabemos si $\varphi_{\mathbf{x}}(\mathbf{x})\uparrow$ ó $\varphi_{\mathbf{x}}(\mathbf{x})\downarrow$), y por tanto no tendremos manera de definir $\Psi(\mathbf{x})$ de forma que sea diferente en ese punto.

Dado que en el segundo caso no podremos hacer que las funciones Ψ y $\varphi_{\mathbf{x}}$ se diferencien localmente en el punto **x** ni en ningún otro, tendremos que definir Ψ de forma que se diferencie globalmente de $\varphi_{\mathbf{x}}$. Haremos que Ψ cumpla las restricciones que impone el conjunto A, es decir que converja para todos los pares y únicamente para ellos. Por tanto, si **x** es impar la función Ψ divergerá, y si **x** es par deberá converger. El valor que devuelva para las palabras pares no podrá ser cualquiera, pues, como ya hemos comentado, también debemos preocuparnos de que Ψ sea diferente localmente de las funciones que pertenecen al conjunto A en alguna entrada.

- Si **x** es par y $\mathbf{x}\in$A, sabemos que $\varphi_{\mathbf{x}}(\mathbf{x})\downarrow$ y que $\Psi(\mathbf{x})$ deberá ser también convergente y diferente de $\varphi_{\mathbf{x}}(\mathbf{x})$, por ejemplo $\varphi_{\mathbf{x}}(\mathbf{x})+3$.

- Si **x** es par y $\mathbf{x}\notin$A, no sabemos si $\varphi_{\mathbf{x}}(\mathbf{x})\uparrow$ ó $\varphi_{\mathbf{x}}(\mathbf{x})\downarrow$, pero $\Psi(\mathbf{x})$ debe converger por la restricción global que nos hemos impuesto. Eso sí, puede producir cualquier valor, sin preocuparnos de $\varphi_{\mathbf{x}}(\mathbf{x})$, porque Ψ tendrá sus índices en el conjunto A, y $\varphi_{\mathbf{x}}$ no.

Todavía tenemos un problema: ¿cómo conseguir que la función Ψ sea diferente de las funciones con índice **x** impar pero que está dentro del conjunto A?. Deberíamos

distinguirnos localmente de ellas, pero no podemos. Al ser $\mathbf{x}\in A$ tenemos que $\varphi_{\mathbf{x}}$ divergerá sobre el propio valor impar **x**. Pero la restricción global obliga a Ψ a diverger también sobre dicho valor impar, con lo cual estamos impotentes para diferenciar ambas funciones en dicho punto.

En definitiva, la dificultad consiste en que sólo tenemos libertad para conformar Ψ sobre los datos pares, pero al mismo tiempo tenemos que tener en cuenta todos los programas, tanto los de código par como impar (pues tanto unos como otros pueden estar en A). Para cumplir con ambas necesidades, en lugar de tener en cuenta la entrada **x** en la función $\varphi_{\mathbf{x}}$, tomaremos la palabra **x** (par) como posible punto para distinguir Ψ de la función $\varphi_{\mathbf{x}/2}$, y por tanto, en lugar de la diagonal principal utilizaremos otra con pendiente 2.

Teniendo en cuenta el esquema anterior, definiremos la función $\Psi: \Sigma^* \dashrightarrow \Sigma^*$ de la siguiente manera:

$$\psi(x) \cong \begin{cases} \varphi_{x/2}(x)+3 & x \bmod 2 = 0 \ \wedge \ x/2 \in A \\ 3 & x \bmod 2 = 0 \ \wedge \ x/2 \notin A \\ \perp & x \bmod 2 = 1 \end{cases}$$

Esta función es computable, como lo demuestra el siguiente programa:

if X1 mod 2 = 0 **then**

  if C_A(X1 / 2) **then** X0 := Φ(X1 / 2, X1)+3; **else** X0 := 3; **end if;**

else X0 := ⊥⊥; **end if**;

Una vez expandidas sus macros, este programa se convertirá en un programa-while con un cierto código **e**, por lo que $\Psi \approx \varphi_{\mathbf{e}}$. Dado que la función converge para todos los pares y diverge con las entradas impares, se tiene que $\mathbf{e}\in A$.

Veamos cuál es el comportamiento de la función Ψ sobre el dato $2*\mathbf{e}$. Es una palabra par y además $(2*\mathbf{e})/2 \in A$, por lo que la única posibilidad es:

$$\varphi_{\mathbf{e}}(2*\mathbf{e}) \overset{\Psi \text{ comput.}}{=} \Psi(2*\mathbf{e}) \overset{\text{def.}\Psi}{=} \varphi_{2*\mathbf{e}/2}(2*e) + 3 = \varphi_e(2*e) + 3 \Rightarrow \varphi_e(2*e)\uparrow$$

que es claramente una contradicción, pues $\varphi_{\mathbf{e}}(2*\mathbf{e})$ es convergente (por ser $\mathbf{e}\in A$).

Hemos encontrado un dato par $(2*\mathbf{e})$ sobre el que Ψ debe producir un resultado distinto de sí mismo. Todo el razonamiento que hemos seguido es correcto, luego la contradicción viene del hecho de suponer que A es decidible. Por tanto concluimos que A no es decidible.

		$\in A$	$\in A$	$\notin A$	$\in A$	$\notin A$	$\in A$	$\notin A$
x	ψ	φ_0	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6
0	$\varphi_0(0)+3$	↓	↓	?	↓	?	↓	?
1	$\perp$	↑	↑	?	↑	?	↑	?
2	$\varphi_1(2)+3$	↓	↓	?	↓	?	↓	?
3	$\perp$	↑	↑	?	↑	?	↑	?
4	3	↓	↓	?	↓	?	↓	?
5	$\perp$	↑	↑	?	↑	?	↑	?
6	$\varphi_3(6)+3$	↓	↓	?	↓	?	↓	?
7	$\perp$	↑	↑	?	↑	?	↑	?
8	3	↓	↓	?	↓	?	↓	?
9	$\perp$	↑	↑	?	↑	?	↑	?
10	$\varphi_5(10)+3$	↓	↓	?	↓	?	↓	?

Siguiendo con la reflexión sobre la aplicación de la diagonalización, consideremos por un momento el conjunto $B = \{ x \in \Sigma^* : W_x = \{y : y \bmod 2 = 0\} = R_x \}$ que tampoco es decidible y veamos hasta qué punto nos serviría la construcción anterior para probarlo por dicha técnica.

La función que tendríamos que definir en este caso debería tener sus índices en el conjunto B para solucionar el mismo problema que en el caso anterior: conseguir diferenciarla globalmente de las funciones cuyos índices no están en B. La función Ψ que hemos construido cumple la primera restricción de B (su dominio es el conjunto de todos los pares), pero en cambio en el rango podemos tener tanto pares como impares,

por lo que su índice **e** no pertenecerá al conjunto B y no conseguiremos nuestro propósito. La función diagonal que deberíamos construir para este caso debería controlar que todos los pares y sólo los pares estuvieran en su rango, por ejemplo distribuyendo los resultados pares de manera ordenada entre los datos pares.

La función $\zeta(x) = \begin{cases} 2 & 2*x \in R_x \\ 8*x & \text{c.c.} \end{cases}$ no es computable

Suponemos que ζ es computable y definiremos una función diagonal computable que a su vez será diferente de todas las funciones computables.

Si ζ es computable podemos distinguir, entre todos los programas, aquellos para los que la función ζ toma el valor 2 de aquellos para los que toma otro valor:

- Si $\zeta(\mathbf{x}) \neq 2$ sabemos que $2*\mathbf{x}$ no está en el rango de $\varphi_{\mathbf{x}}$, por tanto que ninguna entrada produce como imagen el valor $2*\mathbf{x}$. Para que la función Ψ que vamos a construir sea diferente de esta función $\varphi_{\mathbf{x}}$, haremos que $2*\mathbf{x}$ sí esté en el rango de Ψ. Por ejemplo, para la entrada **x** la función Ψ devolverá $2*\mathbf{x}$.

- Cuando $\zeta(\mathbf{x}) = 2$ sabemos que $2*\mathbf{x}$ está en el rango de $\varphi_{\mathbf{x}}$, por tanto existe alguna entrada **y**, aunque no sabemos cuál, tal que $\varphi_{\mathbf{x}}(\mathbf{y}) = 2*\mathbf{x}$. Para que nuestra función Ψ sea diferente de ésta tenemos que conseguir que no devuelva $2*\mathbf{x}$ para ninguna entrada. Esto es compatible con la decisión anterior, ya que los valores de **x** tales que $\zeta(\mathbf{x}) = 2$ no pueden coincidir (y por tanto, sus dobles tampoco) con ningún valor de **x** tal que $\zeta(\mathbf{x}) = 8*\mathbf{x} \neq 2$. Para el dato **x** nos bastará con dejar la función indefinida, ya que sabemos que para ninguna otra entrada se va a producir el resultado concreto $2*\mathbf{x}$.

		$\zeta(0)=2$	$\zeta(1)=8$	$\zeta(2)=2$	$\zeta(3)=2$	$\zeta(4)=32$	
		$0 \in R_0$	$2 \notin R_1$	$4 \in R_2$	$6 \in R_3$	$8 \notin R_4$	
x	Ψ	φ_0	φ_1	φ_2	φ_3	φ_4	
0	$\perp$	0?	$\neg(\varphi_1(0)=2)$	4?	6?	$\neg(\varphi_4(0)=8)$	
1	$2*$	0?	$\neg(\varphi_1(1)=2)$	4?	6?	$\neg(\varphi_4(1)=8)$	
2	$\perp$	0?	$\neg(\varphi_1(2)=2)$	4?	6?	$\neg(\varphi_4(2)=8)$	
3	$\perp$	0?	$\neg(\varphi_1(3)=2)$	4?	6?	$\neg(\varphi_4(3)=8)$	…
4	$8*$	0?	$\neg(\varphi_1(4)=2)$	4?	6?	$\neg(\varphi_4(4)=8)$	
…			…				

La idea queda expresada en la tabla-ejemplo anterior.

Utilizamos la construcción mencionada para hacer la demostración. Supongamos que ζ es computable. Entonces la siguiente función será computable como muestra el programa que figura a continuación:

$$\psi(x) \cong \begin{cases} 2*x & \zeta(x) \neq 2 \\ \perp & \zeta(x) = 2 \end{cases}$$

if $\zeta(X1) \neq 2$ **then** X0 := 2*X1 ; **else** X0 := $\perp\!\!\!\perp$; **end if**;

El programa anterior, una vez expandidas sus macros y codificando el programa while resultante tendrá un cierto índice **e** tal que $\Psi \cong \varphi_e$.

Podemos preguntarnos si $2*\mathbf{e}$ será o no un elemento de R_e y veremos que ninguna de las dos alternativas es posible:

$$2*\mathbf{e} \notin R_{\mathbf{e}} \Rightarrow \zeta(\mathbf{e}) = 8*\mathbf{e} \neq 2 \overset{\text{def. } \psi}{\Rightarrow} \Psi(\mathbf{e}) = 2*\mathbf{e} \overset{\psi \cong \varphi_e}{\Rightarrow} \varphi_{\mathbf{e}}(\mathbf{e}) = 2*\mathbf{e} \Rightarrow 2*\mathbf{e} \in R_{\mathbf{e}}$$

$$2*\mathbf{e} \in R_{\mathbf{e}} \Rightarrow \zeta(\mathbf{e}) = 2 \overset{\text{def. } \psi}{\Rightarrow} \Psi(\mathbf{e}) \uparrow \Rightarrow 2*\mathbf{e} \notin \text{Ran}(\Psi) \overset{\psi \cong \varphi_e}{\Rightarrow} 2*\mathbf{e} \notin R_{\mathbf{e}}$$

Por tanto se produce una contradicción, que sólo puede venir de la hipótesis de computabilidad de ζ, por lo que queda demostrado que esta última es falsa.

Si considerásemos el conjunto $B = \{ \mathbf{x} \in \Sigma^* : \exists \mathbf{z} \quad 2*\mathbf{z} \in R_{\mathbf{X}} \}$ cuya función característica se parece formalmente a ζ, podríamos inicialmente pensar que la técnica de diagonalización se aplicaría de manera similar, pero sin embargo, podemos ver las dificultades que entrañaría hacerlo para concluir que no siempre va a ser posible aplicar esta técnica.

La función característica del conjunto B aplicada sobre un programa de código **x** no nos da información suficiente en ningún caso acerca de ningún valor local de $\varphi_{\mathbf{x}}$. Por tanto, cuando definimos la función ψ no podemos obligarla a diferenciarse de $\varphi_{\mathbf{x}}$ en ningún punto concreto, así que tendremos que hacerlas diferenciarse globalmente:

- Cuando $x \notin B$ tenemos que nuestra función $\varphi_{\mathbf{x}}$ no tiene ningún par en el rango. Parece razonable que Ψ devuelva en alguna entrada un número par para que las dos funciones sean diferentes.

- Cuando $x \in B$ lo único que sabemos de $\varphi_{\mathbf{x}}$ es que tiene algún par en el rango, pero no sabemos ante qué dato lo produce. Así pues, interesará que la función Ψ no devuelva ningún número par para que sean diferentes.

  Se ve claramente que las dos restricciones son incompatibles.

$A = \{ x: 2*x \in R_x \wedge 2*x \in W_x \}$ no es decidible

Debemos suponer que A es decidible y basándonos en ello construir una función diagonal ψ distinta de todas las funciones computables. Con esta suposición, dado un programa cualquiera de índice **x** podemos preguntar si está en A o no.

- Si $x \notin A$, la información que obtenemos sobre **x** es que, o bien $\varphi_x(2*x)\uparrow$, o bien $2*x$ no está en el rango, pero no sabemos cuál de las dos cosas ocurre (o si ocurren ambas). Para diferenciar ψ de φ_x lo hacemos justamente en el punto $2*x$, haciendo que su imagen sea $2*x$, con lo que en cualquiera de los dos casos será diferente porque introducimos $2*x$ tanto en el dominio como en el rango de ψ.

- Si $x \in A$, sabemos que φ_x converge para $2*x$ y también que $2*x$ está en el rango, aunque no sabemos frente a qué dato se produce. Para conseguir que ψ sea diferente de φ_x será suficiente con hacer que diverja para $2*x$.

Por lo tanto, lo que hacemos no es recorrer exactamente la diagonal, sino la diagonal formada por las entradas pares, ya que lo que ocurra en los lugares impares es indiferente para la resolución del problema. Mejor dicho, no lo es del todo, porque si ψ convergiera para esos valores podríamos introducir elementos indeseables en su rango. Así pues, divergeremos en todos los datos impares. En el gráfico de la siguiente página podemos ver los lugares de la diagonal que recorremos y en los que la nueva función se diferenciará de cada una de las computables.

Utilizamos la construcción mencionada para hacer la demostración. Supongamos que A es decidible. Entonces la siguiente función será computable, como muestra el programa que figura a continuación:

$$\psi(x) \cong \begin{cases} \perp & x \bmod 2 = 1 \\ \perp & x \bmod 2 = 0 \wedge x/2 \in A \\ x & x \bmod 2 = 0 \wedge x/2 \notin A \end{cases}$$

if X1 mod 2 = 0 **and not** C_A(X1 / 2) **then** X0 := X1; **else** X0 := $\perp\!\!\!\perp$; **end if**;

El programa anterior, una vez expandidas sus macros, se convertirá en un programa-while con un cierto índice **e**, de forma que $\psi \cong \varphi_e$. Veamos que tal función no existe, pues su índice no podría estar en A ni en su complementario, ya que no hay definición posible para $\psi(2*e)$:

$$e \in A \overset{\text{def. } \psi}{\Rightarrow} \psi(2*e)\uparrow \overset{\psi \cong \varphi_e}{\Rightarrow} \varphi_e(2*e)\uparrow \Rightarrow 2*e \notin W_e \Rightarrow e \notin A$$

$$\mathbf{e}\notin A \;\overset{\text{def.}\psi}{\Rightarrow}\; \boldsymbol{\psi}(2*\mathbf{e})=2*\mathbf{e} \;\overset{\psi\cong\varphi_e}{\Rightarrow}\; \varphi_{\mathbf{e}}(2*\mathbf{e})=2*\mathbf{e} \;\Rightarrow 2*\mathbf{e}\in W_{\mathbf{e}} \wedge 2*\mathbf{e}\in R_{\mathbf{e}} \Rightarrow \mathbf{e}\in A$$

Tenemos que $e\in A \Leftrightarrow e\notin A$. Esta contradicción viene del hecho de suponer que A es decidible, por lo que queda demostrado que esto último no es posible.

		$\in A$	$\in A$	$\notin A$	$\in A$	$\notin A$
x	ψ	φ_0	φ_1	φ_2	φ_3	φ_4
0	↑	↓0?	2?	°4?	6?	°8?
1	↑	0?	2?	°4?	6?	°8?
2	↑	0?	↓2?	°4?	6?	°8?
3	↑	0?	2?	°4?	6?	°8?
4	4	0?	2?	↑°4?	6?	°8?
5	↑	0?	2?	°4?	6?	°8?
6	↑	0?	2?	°4?	↓6?	°8?
7	↑	0?	2?	°4?	6?	°8?
8	8	0?	2?	°4?	6?	↑°8?

. . .

. . .

$\{ x: \forall y\ \varphi_x(y)\downarrow \wedge R_x \neq \Sigma^* \}$ no es decidible

El conjunto de funciones que define este conjunto son aquellas que verifican dos propiedades simultáneamente: por un lado convergen para cualquier entrada, y por otro lado no todas las posibles imágenes están en su rango, es decir son las funciones totales y no sobreyectivas, por lo que llamaremos al conjunto TNS.

Supongamos que TNS es decidible. Entonces su función característica C_{TNS} es computable. Por tanto, podemos preguntar, dada una función de índice **x**, si éste pertenece o no al conjunto TNS, y basándonos en ello definir una función ψ computable. Para construirla utilizaremos todas las funciones computables y haremos que sea diferente de todas ellas, de forma que la función ψ y cada una de las $\varphi_{\mathbf{x}}$ se diferencien por lo menos en un punto.

- Si **x** está en TNS, sabremos que φ_x es total y no sobreyectiva, por tanto $\varphi_x(\mathbf{x})\downarrow$. Para que $\psi(\mathbf{x})$ sea diferente de dicho valor tendríamos muchas maneras de hacerlo: tomando un valor que modifique de alguna manera el valor de $\varphi_x(\mathbf{x})$, ó haciendo que $\psi(\mathbf{x})\uparrow$.

- Si **x** no está en TNS, sabremos que ó $\varphi_{\mathbf{x}}$ no es total ó es sobreyectiva. Pero no habrá manera de saber qué hace sobre la palabra **x** (si $\varphi_x(\mathbf{x})\downarrow$ ó $\varphi_x(\mathbf{x})\uparrow$), y no podremos definir $\psi(\mathbf{x})$ de forma que sea diferente en esa palabra.

Dado que en el segundo caso no podremos hacer que las funciones ψ y $\varphi_{\mathbf{x}}$ se diferencien localmente en la palabra **x**, tendremos que definir ψ de forma que se diferencie globalmente de $\varphi_{\mathbf{x}}$. Haremos que la función ψ sea total y no sobreyectiva, que cumpla las restricciones que impone el conjunto TNS. Nuestro propósito va a ser que $dom(\psi) = \Sigma^* \neq ran(\psi)$. Lo primero es sencillo, y para conseguir lo segundo basta con excluir cuidadosamente un resultado concreto al construir ψ. Por ejemplo, podemos evitar que la función devuelva 0.

- Si **x** está en TNS, definiremos $\psi(\mathbf{x})$ como $\varphi_{\mathbf{x}}(\mathbf{x})+1$, que será un valor convergente, diferente de 0, y qe permite diferenciar localmente ψ de $\varphi_{\mathbf{x}}$.

- Si **x** no está en TNS, definiremos $\psi(\mathbf{x})$ directamente como un valor diferente de 0, por ejemplo 1, para seguir sosteniendo el objetivo de que ψ sea total pero no contenga el 0 en su rango. Así se diferenciará globalmente de $\varphi_{\mathbf{x}}$.

  Utilizamos este razonamiento para definir la función diagonal:

$$\psi(x) = \begin{cases} \varphi_x(x) + 1 & x \in TNS \\ 1 & x \notin TNS \end{cases}$$

El esquema se refleja en la siguiente tabla-ejemplo:

		$\in$TNS	$\in$TNS	$\notin$TNS	$\notin$TNS	$\in$TNS	$\in$TNS	
		$R_0 \neq \Sigma^*$	$R_1 \neq \Sigma^*$	¿R_2?	¿R_3?	$R_4 \neq \Sigma^*$	$R_5 \neq \Sigma^*$	
x	Ψ	φ_0	φ_1	φ_2	φ_3	φ_4	φ_5	
0	$\varphi_0(0)+1$	↓	↓	?	?	↓	↓	
1	$\varphi_1(1)+1$	↓	↓	?	?	↓	↓	
2	1	↓	↓	?	?	↓	↓	
3	1	↓	↓	?	?	↓	↓	…
4	$\varphi_4(4)+1$	↓	↓	?	?	↓	↓	
5	$\varphi_5(5)+1$	↓	↓	?	?	↓	↓	
				…				

Como suponemos que TNS es decidible entonces la computabilidad de la función quedaría demostrada con el programa:

if $C_{TNS}(X1)$ **then** X0:= $\Phi(X1, X1)+1$; **else** X0:=1; **end if**;

El programa anterior, una vez expandidas sus macros y codificado el programa-while resultante, tendrá un cierto índice **e** tal que $\psi = \varphi_{\mathbf{e}}$. La función ψ es claramente total, ya que siempre converge, y no sobreyectiva, pues $0 \notin \text{ran}(\psi)$, luego $\mathbf{e} \in$ TNS. Pero si intentamos aplicar la función sobre su propio índice:

$$\mathbf{e} \in \text{TNS} \overset{\text{def. } \psi}{\Rightarrow} \varphi_{\mathbf{e}}(\mathbf{e}) = \psi(\mathbf{e}) = \varphi_{\mathbf{e}}(\mathbf{e})+1 \Rightarrow \varphi_{\mathbf{e}}(\mathbf{e})\uparrow$$

con lo que llegamos a una contradicción, ya que la función que hemos definido es total. Esta contradicción proviene del hecho de haber supuesto que TNS es decidible, por lo que queda demostrado que esto último no es posible.

$$\zeta(x) = \begin{cases} 2*x & \varphi_x \textbf{ es total} \\ & \textbf{y sobreyectiva} \\ 2*x+1 & \textbf{c.c.} \end{cases} \quad \textbf{no es computable}$$

Supongamos que ζ es computable. Usando la técnica de diagonalización y basándonos en la computabilidad de ζ, podemos encontrar una función ψ computable, pero que no coincida con ninguna de las funciones computables, llegando así a una contradicción. Para ello, haremos que ψ sea total y sobreyectiva, pero diferente de todas las totales y sobreyectivas.

Para que sea total, hemos de asignar un valor en cada punto (esto es fácil), y para que sea sobreyectiva hemos de asegurar que todos los naturales son asignados alguna vez (esto es lo más difícil). Una manera de hacerlo es asignar los naturales en orden, siempre que sea posible: primero el 0, luego el 1, después el 2, etc., intentando siempre colocar el primer valor que aún no hemos asignado, y en caso de que no se pueda (porque necesitamos diferenciarnos localmente de una cierta $\varphi_{\mathbf{x}}$ total y sobreyectiva en el punto **x**), el segundo que tengamos libre. Uno de los dos será siempre posible asignarlo.

- Si $\zeta(\mathbf{x})$ es impar, entonces la función $\varphi_{\mathbf{x}}$ no es total y sobreyectiva, y ψ va a ser distinta de $\varphi_{\mathbf{x}}$ globalmente, por lo que asignamos a $\psi(\mathbf{x})$ el menor valor aún no atribuido.

- Si $\zeta(\mathbf{x})$ es par, entonces la función $\varphi_{\mathbf{x}}$ es total y sobreyectiva, y ψ va a ser distinta de $\varphi_{\mathbf{x}}$ en el punto **x**. Comparamos el menor valor aún no asignado con $\varphi_{\mathbf{x}}(\mathbf{x})$, que existe por ser $\varphi_{\mathbf{x}}$ total. Si son distintos éste será el valor de $\psi(\mathbf{x})$, y si coinciden buscamos el siguiente valor aún no asignado.

De esta forma siempre asignamos un valor (ψ será total), y finalmente asignamos todos los resultados posibles (ψ será sobreyectiva). Formalmente, la función que estamos definiendo es:

$$\psi(0) = \begin{cases} 0 & \zeta(0) \bmod 2 \neq 0 \\ 0 & \zeta(0) \bmod 2 = 0 \wedge \varphi_0(0) \neq 0 \\ 1 & \zeta(0) \bmod 2 = 0 \wedge \varphi_0(0) = 0 \end{cases}$$

$$\psi(x) = \begin{cases} \mu z(\forall y < x(\psi(y) \neq z) \wedge \varphi_x(x) \neq z) & \zeta(x) \bmod 2 = 0 \\ \mu z(\forall y < x(\psi(y) \neq z)) & \zeta(x) \bmod 2 \neq 0 \end{cases}$$

donde $\mu\mathbf{z}P(\mathbf{z})$ significa el menor z que verifica el predicado $P(\mathbf{z})$.

Según hemos explicado la función ψ sigue el esquema de diagonalización que ilustramos en la siguiente tabla-ejemplo:

		ζ(0) par	ζ(1) impar	ζ(2) par	ζ(3) par	ζ(4) impar	ζ(5) par	ζ(6) impar	
x	**ψ**	**φ_0**	**φ_1**	**φ_2**	**φ_3**	**φ_4**	**φ_5**	**φ_6**	
0	**1**	0	?	↓	↓	?	↓	?	
1	**0**	↓	?	↓	↓	?	↓	?	
2	**3**	↓	?	2	↓	?	↓	?	
3	**4**	↓	?	↓	2	?	↓	?	…
4	**2**	↓	?	↓	↓	?	↓	?	
5	**5**	↓	?	↓	↓	?	7	?	
6	**6**	↓	?	↓	↓	?	↓	?	

…

La computabilidad de esta función ψ se basa en la de ζ, y queda demostrada con el siguiente programa:

```
PRIMER:=0; SEGUN:=1; AUX:=0;
while AUX ≤ X1 loop
    if ζ(AUX) mod 2 ≠ 0
    then  X0:= PRIMER; PRIMER:= SEGUN; SEGUN:= SEGUN+1;
    else  Z:= Φ(AUX, AUX);
          if Z ≠ PRIMER
          then X0:= PRIMER; PRIMER:= SEGUN; SEGUN:= SEGUN+1;
          else X0:= SEGUN;  SEGUN:= SEGUN+1;
          end if;
    end if;
    AUX := AUX +1;
end loop;
```

Pero por otro lado esta función es distinta de todas las computables por construcción, y ello conduce a la contradicción que veremos a continuación.

Existe un índice **e** tal que $\psi \cong \varphi_{\mathbf{e}}$. Además esta función es biyectiva por construcción, luego $\zeta(\mathbf{e}) \bmod 2 = 0$. Si $v = \mu z(\forall y{<}\mathbf{e}\ (\psi(y) \neq z) \wedge \varphi_{\mathbf{e}}(\mathbf{e}) \neq z)$ entonces $\psi(\mathbf{e}) = v$ por definición de la función ψ . Luego llegamos a que $\psi(\mathbf{e}) = v \Leftrightarrow \varphi_{\mathbf{e}}(\mathbf{e}) \neq v$, es decir, $\varphi_{\mathbf{e}}(\mathbf{e}) = v \Leftrightarrow \varphi_{\mathbf{e}}(\mathbf{e}) \neq v$. Contradicción provocada por el hecho de suponer que ζ era computable, por lo que queda demostrado que esto último no es posible.

Este es, además, uno de los esquemas que se suelen utilizar para demostrar que el conjunto de los índices de funciones computables biyectivas (PERM) no es decidible porque la función ψ además de ser total y sobreyectiva es inyectiva, aunque esto último para nuestro caso no era necesario.

$A = \{ x: \varphi_x$ es oscilante $\}$ no es decidible

Decimos que una función η es oscilante cuando:

I) η es total

II) siempre se da que $\eta(x) \neq \eta(x+1)$

III) no se verifica para ningún dato x que $\eta(x) > \eta(x+1) > \eta(x+2)$

IV) no se verifica para ningún dato x que $\eta(x) < \eta(x+1) < \eta(x+2)$

Es decir, cuando tiene la propiedad de que los valores que va produciendo para datos consecutivos son oscilantes (crecen y decrecen a cada paso).

Para utilizar el método de diagonalización hemos de suponer que A es decidible y construir una función **g** distinta de todas las funciones computables, pero que podría ser computada bajo dicha hipótesis. Imaginemos que A es decidible, y que, por consiguiente, su función característica C_A es computable.

Sea **x** el código de un programa cualquiera. En principio queremos que $\mathbf{g}(\mathbf{x})$ y $\varphi_{\mathbf{x}}(\mathbf{x})$ sean diferentes, y para conseguirlo preguntamos si $C_A(\mathbf{x})$.

- Si la respuesta es positiva, la cosa resulta sencilla: $\varphi_{\mathbf{x}}$ es oscilante, y por tanto total, por lo que sabemos que $\varphi_{\mathbf{x}}(\mathbf{x})\downarrow$, y en caso de necesidad podemos calcular su valor para conseguir que $\mathbf{g}(\mathbf{x})$ sea diferente.

- Sin embargo, si la respuesta es negativa la información que obtenemos sobre $\varphi_{\mathbf{x}}$ es muy pobre, y en particular no averiguamos nada en absoluto ni sobre $\varphi_{\mathbf{x}}(\mathbf{x})$ ni sobre ningún otro valor concreto. Esto nos lleva a adoptar la estrategia de diferenciar $\mathbf{g}(\mathbf{x})$ de las funciones computables no oscilantes de manera global, es decir, haciendo que **g** sí sea oscilante.

Para ello, haremos lo siguiente: en general, la función **g** oscilará entre los valores 0 (datos pares) y 10 (datos impares). No obstante, si en algún momento no puede devolver esos valores porque necesita distinguirse localmente de alguna función computable oscilante, utilizará de forma alternativa el 1 y el 9 respectivamente. La definición de **g** quedaría como:

$$g(x) = \begin{cases} 0 & x \bmod 2 = 0 \wedge (x \notin A \vee (x \in A \wedge \varphi_x(x) \neq 0)) \\ 1 & x \bmod 2 = 0 \wedge x \in A \wedge \varphi_x(x) = 0 \\ 9 & x \bmod 2 \neq 0 \wedge x \in A \wedge \varphi_x(x) = 10 \\ 10 & x \bmod 2 \neq 0 \wedge (x \notin A \vee (x \in A \wedge \varphi_x(x) \neq 10)) \end{cases}$$

Por la decidibilidad de A, la función **g** es computada por el programa:

```
if X1 mod 2 = 1 then
    X0 := 10;
    if C_A(X1) then
        R := Φ(X1, X1);
        if R=10 then X0 := 9; end if;
    end if;
else X0 := 0;
    if C_A(X1) then
        R := Φ(X1, X1);
        if R=0 then X0 := 1; end if;
    end if;
end if;
```

Por ser computable, debe existir un código **e,** correspondiente al programa-while resultante tras la expansión de las macros del programa anterior, que verifica $\mathbf{g} = \varphi_{\mathbf{e}}$. Por ser oscilante, tenemos que $\mathbf{e} \in A$. Para estudiar la naturaleza del valor de $\varphi_{\mathbf{e}}(\mathbf{e})$ vamos a distinguir dos casos: que **e** sea par o que sea impar.

Si **e** es par:

$$\left.\begin{array}{r} \varphi_e(\mathbf{e}) = 0 \\ \mathbf{e} \in A \\ \mathbf{e} \bmod 2 = 0 \end{array}\right\} \overset{\text{def. g}}{\Rightarrow} g(\mathbf{e}) = 1 \overset{g=\varphi_e}{\Rightarrow} \varphi_e(\mathbf{e}) = 1 \Rightarrow \varphi_e(\mathbf{e}) \neq 0$$

$$\left.\begin{array}{r} \varphi_e(\mathbf{e}) \neq 0 \\ \mathbf{e} \in A \\ \mathbf{e} \bmod 2 = 0 \end{array}\right\} \overset{\text{def. g}}{\Rightarrow} g(\mathbf{e}) = 0 \overset{g=\varphi_e}{\Rightarrow} \varphi_e(\mathbf{e}) = 0$$

Si **e** es impar:

$$\left.\begin{array}{r} \varphi_e(\mathbf{e}) = 10 \\ \mathbf{e} \in A \\ \mathbf{e} \bmod 2 \neq 0 \end{array}\right\} \overset{\text{def. g}}{\Rightarrow} g(\mathbf{e}) = 9 \overset{g=\varphi_e}{\Rightarrow} \varphi_e(\mathbf{e}) = 9 \Rightarrow \varphi_e(\mathbf{e}) \neq 10$$

$$\left.\begin{array}{r} \varphi_e(\mathbf{e}) \neq 10 \\ \mathbf{e} \in A \\ \mathbf{e} \bmod 2 \neq 0 \end{array}\right\} \overset{\text{def. g}}{\Rightarrow} g(\mathbf{e}) = 10 \overset{g=\varphi_e}{\Rightarrow} \varphi_e(\mathbf{e}) = 10$$

Tenemos que si **e** es par, se llega a una contradicción, y si es impar se llega a otra. La función **g** está bien definida pero no puede asignar ningún valor a **e**, por lo que nos encontramos con que nuestra suposición inicial de que A era decidible no es factible.

$\overline{K} = \{ x: \varphi_x (x)\uparrow \}$ no es semidecidible

La semidecibilidad de este conjunto depende de la computabilidad de su función semicaracterística:

$$\chi_{\overline{K}}(x) \cong \begin{cases} \text{true} & x \in \overline{K} \\ \perp & \text{c.c.} \end{cases}$$

Alternativamente podríamos haber puesto como condición $x \notin W_x$ ó $\varphi_x(x)\uparrow$, por ejemplo:

$$\chi_{\overline{K}}(x) \cong \begin{cases} \text{true} & \varphi_x(x)\uparrow \\ \perp & \varphi_x(x)\downarrow \end{cases}$$

Suponemos que el conjunto $\overline{K}$ es semidecidible, y por tanto que su función semicaracteística es computable. Basándonos en ello podemos definir, por diagonalización, una función ψ computable y diferente de todas las funciones computables. Haremos que la función ψ y cada una de las funciones computables $\varphi_{\mathbf{x}}$ se diferencien por lo menos en un punto, el de la diagonal **x**.

- Si $\mathbf{x} \in \overline{K}$, la función semicaracterística de $\overline{K}$ aplicada sobre **x** converge. Para que la función ψ sea diferente, y además en este punto, basta con que converja a su vez.

- Si $\mathbf{x} \notin \overline{K}$, la función semicaracterística de $\overline{K}$ aplicada sobre **x** diverge. Esto quiere decir que el mero hecho de indagar si $\chi_{\overline{K}}(\mathbf{x})$ se cumple obliga a ciclar. Afortunadamente, para que la función ψ sea diferente de φ_x en este punto nos basta con diverger, que en este caso es lo único que podemos hacer.

El esquema quedaría ilustrado en la tabla-ejemplo de la página siguiente donde podemos ver que la nueva función se diferencia de cada función $\varphi_{\mathbf{x}}$ en el punto de la diagonal. La definición formal de la función es como sigue:

$$\psi(x) \cong \begin{cases} 50 & \chi_{\overline{K}}(x)\downarrow \\ \perp & \chi_{\overline{K}}(x)\uparrow \end{cases}$$

Suponiendo que $\overline{K}$ es semidecidible tenemos que la función que hemos definido es computada por el siguiente programa:

```
X0 := χ_K̄ (X1);
X0 := 50;
```

		$\notin \overline{K}$	$\in \overline{K}$	$\notin \overline{K}$	$\in \overline{K}$	$\notin \overline{K}$	$\notin \overline{K}$	$\in \overline{K}$	
x	ψ	φ_0	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	...
0	↑	↓	?	?	?	?	?	?	
1	**50**	?	↑	?	?	?	?	?	
2	↑	?	?	↓	?	?	?	?	
3	**50**	?	?	?	↑	?	?	?	...
4	↑	?	?	?	?	↓	?	?	
5	↑	?	?	?	?	?	↓	?	
6	**50**	?	?	?	?	?	?	↑	

...

Una vez expandidas sus macros este programa se convertirá en un programa while con un cierto índice **e**, es decir, $\psi \cong \varphi_{\mathbf{e}}$. Veamos cuál es el comportamiento de la función sobre su propio índice **e**:

$$\varphi_e(\mathbf{e})\uparrow \Rightarrow \psi(\mathbf{e})\uparrow \Rightarrow \chi_{\overline{K}}(\mathbf{e})\uparrow \Rightarrow \mathbf{e} \notin \overline{K} \Rightarrow \varphi_e(\mathbf{e})\downarrow$$

$$\varphi_e(\mathbf{e})\downarrow \Rightarrow \psi(\mathbf{e})\downarrow \Rightarrow \chi_{\overline{K}}(\mathbf{e})\downarrow \Rightarrow \mathbf{e} \in \overline{K} \Rightarrow \varphi_e(\mathbf{e})\uparrow$$

La función ψ en el punto **e** no puede ni converger ni diverger, por lo que llegamos a una contradicción.

Todo el razonamiento que hemos seguido es correcto, luego la contradicción viene del hecho de suponer que $\overline{K}$ es semidecidible, por lo que queda demostrado que esto no es posible.

$$\xi(x) \cong \begin{cases} x^3 & \varphi_x \text{ es creciente} \\ \perp & \text{c.c.} \end{cases} \qquad \text{no es computable}$$

Decimos que una función η es creciente cuando:

I) η es total

II) para datos y, z tales que y>z siempre se verifica que $\eta(y) > \eta(z)$

Es decir, cuando tiene la propiedad de que los valores que va produciendo van siendo cada vez mayores. Por ejemplo la identidad es una función creciente.

No podemos usar el método de diagonalización de la misma forma que en el caso anterior. A la hora de construir ψ diferente de todas las funciones computables basándonos en ξ, ocurre que $\xi(\mathbf{x})$ diverge en el momento más inoportuno: cuando $\varphi_{\mathbf{x}}$ es no creciente. Esto quiere decir que, a la imposibilidad de diferenciarnos localmente de $\varphi_{\mathbf{x}}$, porque el hecho de no ser creciente no revela nada sobre ningún punto concreto de la misma, se nos añade la imposibilidad de diferenciarnos globalmente de $\varphi_{\mathbf{x}}$, porque no podemos conseguir que nuestra función diagonal sí sea creciente al estar obligados a diverger en el punto **x** (no olvidemos que creciente implica total).

El problema es que la función ξ, cuyo dominio está formado por los índices de funciones computables crecientes, no es aprovechable para la diagonalización. Sin embargo es posible atacar el problema de otra manera con otra función cuyo rango esté formado por los índices de funciones computables crecientes. Esta nueva función nos permitirá realizar el argumento de diagonalización sobre las funciones computables crecientes (mucho más cómodas de manipular por ser totales), en lugar de sobre todas las computables.

El vehículo que nos permitirá pasar de una a otra será el teorema de caracterización de los conjuntos semidecidibles. Supongamos que ξ es computable. Su dominio está formado por los índices de funciones computables crecientes, y lo llamaremos CRE. Como ξ es computable, dom(ξ) = CRE es un conjunto semidecidible. Dado que CRE es no vacío (al menos contiene todos los programas que computan la función identidad), el teorema de caracterización nos asegura que CRE coincide con el rango de una función total y computable **g**, es decir, que CRE = ran(g) = {g(0), g(1), g(2), …, g(i), …}. Por tanto, todo elemento $\mathbf{i} \in$ CRE puede ponerse de la forma **g(j)** para algún **j**.

La ventaja que tenemos ahora es que disponemos de una enumeración $\{\varphi_{g(0)}, \varphi_{g(1)}, \varphi_{g(2)} \ldots\}$ de todas las funciones computables crecientes. Trataremos entonces

de construir por diagonalización una función computable y creciente diferente a todas ellas.

Fijémonos en la tabla-ejemplo. Dado que cada una de las funciones que aparecen en las cabeceras de columna son crecientes y por tanto totales, por lo que podemos averiguar el valor de cualquiera de dichas funciones en la diagonal y construir nuestra función diagonal **h** de forma que se diferencie de cada una de ellas en ese punto, por ejemplo sumando 1 a ese valor. Pero además queremos que **h** sea creciente, para lo cuál podemos sumar además el valor asignado por la función en el punto anterior. El esquema explicado y reflejado en la tabla nos lleva a la siguiente definición inductiva:

$$h(0) = 1 + \varphi_{g(0)}(0)$$

$$h(n+1) = h(n) + 1 + \varphi_{g(n+1)}(n+1)$$

que es claramente creciente.

x	h	$\varphi_{g(0)}$	$\varphi_{g(1)}$	$\varphi_{g(2)}$	$\varphi_{g(3)}$	$\varphi_{g(4)}$	…
0	$1+\varphi_{g(0)}(0)$	↓	↓	↓	↓	↓	
1	$g(0)+\varphi_{g(1)}(1)+1$	↓	↓	↓	↓	↓	
2	$g(1)+\varphi_{g(2)}(2)+1$	↓	↓	↓	↓	↓	
3	$g(2)+\varphi_{g(3)}(3)+1$	↓	↓	↓	↓	↓	…
4	$g(3)+\varphi_{g(4)}(4)+1$	↓	↓	↓	↓	↓	
5	$g(4)+\varphi_{g(5)}(5)+1$	↓	↓	↓	↓	↓	
		…					

El programa que computaría a **h** sería el siguiente:

AUX := 0; X0 := 0;

while AUX ≤ X1 **loop**

X0 := X0 + Φ(g(AUX), AUX) + 1;

AUX := AUX +1;

end loop;

Por ser **h** computable existirá un índice e tal que $\mathbf{h} \cong \varphi_{\mathbf{e}}$. Como además es creciente, tenemos que $\mathbf{e} \in CRE = ran(\mathbf{g})$, por lo que debe existir un valor **a** que verifica $\mathbf{e} = \mathbf{g}(\mathbf{a})$, y por tanto $\mathbf{h} \cong \varphi_{\mathbf{g(a)}}$. Pero veamos qué ocurre si aplicamos la función en el punto **a**:

Si **a**=0

$$\varphi_{\mathbf{g(0)}}(0) \cong \mathbf{h}(0) \cong 1 + \varphi_{\mathbf{g(0)}}(0)$$

Si **a**>0

$$\varphi_{\mathbf{g(a)}}(\mathbf{a}) \cong \mathbf{h}(\mathbf{a}) \cong \mathbf{h}(\mathbf{a}\text{-}1) + 1 + \varphi_{\mathbf{g(a)}}(\mathbf{a})$$

En ambos casos se obtienen igualdades que sólo pueden cumplirse si $\varphi_{\mathbf{g(a)}}(\mathbf{a})\uparrow$, pero eso contradice el hecho de que $\mathbf{h} \cong \varphi_{\mathbf{g(a)}}$ es creciente, y por tanto total. La contradicción se debe a la suposición de que ξ es computable, por lo que queda demostrado que no lo es.

Referencias

[Har 87] D. HAREL. Algorithmics. The spirit of Computing. Addison-Wesley, 1987

[IIS 96] J. IBAÑEZ; A. IRASTORZA; A. SANCHEZ. *Los Programas while. Bases para una teoría de la computabilidad.* Informe interno UPV/EHU/LSI/TR 5-96.

[MAK 88] R. N. MOLL; M. A. ARBIB; A. J. KFOURY. A programming approach to computablity. Springer-Verlag, 1988

[SW 88] R. SOMMERHALDER; S. C. van WESTRHENEN The theory of computability. Programs, Machines, Effectiveness and Feasibility. Addison-Wesley 1.988

[Sta 81] G. STAHL. *El método diagonal en teoría de conjuntos.* Teorema. Vol 11, nº1, pp 27-35, 1981