COMPUTABILIDAD, COMPLEJIDAD Y TEOREMAS DE GÖDEL

Guillermo Martínez

Lo que sigue son las notas correspondientes a un curso de tres clases dictado en la Universidad Nacional del Sur, en abril de 1999, durante el V Congreso Antonio Monteiro. El propósito de ese curso era dar una aproximación novedosa, vía la paradoja de Berry, a los teoremas clásicos de incompletitud de Gödel. Esta aproximación, concebida por primera vez por Chaitin como punto de partida de su teoría de complejidad algorítmica, y formulada luego independientemente por Boolos y Caicedo, tiene también puntos de contacto con los trabajos de Turing sobre el Halting Problem y con la teoría de Smullyan de sistemas formales abstractos. Quise organizar por eso el curso de este modo: la primera parte corresponde a una introducción histórica al problema de la incompletitud de la Aritmética y debe gran parte del enfoque a la primera conferencia de G. Chaitin en su libro *The limits of mathematics* (2); se definen las principales nociones de teoría completa, teoría axiomática, etc. y se formula el Teorema de Incompletitud de Gödel, junto con la demostración informal de Turing sobre la indecidibilidad del Halting Problem.

En la segunda parte se da una demostración rigurosa de la indecidibilidad del Halting Problem, pero en vez de utilizar la noción de Máquinas de Turing, lo hacemos mediante una definición muy simple de *programa,* siguiendo los lineamientos de M. Davis en (5). Esto nos permite hacer una conexión inmediata con la teoría de complejidad descriptiva y la noción de programas elegantes de Chaitin.

En la tercera parte, siguiendo la exposición de (8) sobre los resultados de Chaitin, probamos que hay un número real entre 0 y 1, que codifica completamente el Halting Problem, en el sentido de que la información sobre cuáles son los primeros n dígitos de este número equivale a tener un método de decisión para determinar a priori si un programa de longitud menor o igual que n se detendrá o no.

Finalmente la cuarta parte correspondería a la última clase del curso, en la que expuse la demostración del Teorema de Incompletitud de Gödel vía la paradoja de Berry desarrollada por X. Caicedo en (1). Pero como el trabajo de Caicedo está en castellano, y escrito de una manera admirable, me exime de agregar aquí más nada.

Quiero agradecer al profesor Manuel Abad y a los organizadores del congreso la invitación para dar estas charlas, así como a todos los asistentes que siguieron el curso. En particular agradezco muchísimo la ayuda invalorable de la profesora Marta Sagastume, que me ofreció su colaboración y muchas horas de su tiempo para la redacción definitiva de estas notas. Realmente, nunca las hubiera terminado sin la paciencia y el apoyo constante de Marta.

PARTE 1: El programa de Hilbert.

Hacia 1900, en la tradición de los cinco postulados de Euclides y de los principios de la lógica simbólica esbozados por **L**eibniz, David Hilbert propuso como programa para el nuevo siglo la formulación de un sistema axiomático general para toda la matemática. Por un lado, esto debía evitar las paradojas que habían puesto en crisis por ejemplo a la teoría de conjuntos y por otro lado establecer las bases para que la verdad de cualquier proposición matemática pudiera comprobarse de una manera mecánica y objetiva, esto es, que una vez propuesta una tesis, la corroboración de esa tesis no dependiera de la inteligencia o intuición humana. Nótese que detrás de este programa está el convencimiento de que todo enunciado matemático verdadero lo es por alguna razón, que puede explicitarse. Esta razón, compartible y comprobable, es lo que los matemáticos llaman *prueba*. En la práctica matemática usual, esto corresponde a dos momentos en general separados: la aprehensión de una noción verdadera y en una segunda etapa la búsqueda de los argumentos que demuestran esta verdad (compárese con la noción de culpabilidad y evidencia en la justicia).

Los dos principales requisitos que pedía Hilbert para este sistema axiomático general eran:

1. **Consistencia:** Esto significa que no pueden deducirse como teoremas del sistema una fórmula **A** y su negación $\neg$**A.** Si esto ocurre, a partir del principio del tercero excluído pueden obtenerse como teoremas absolutamente todas las proposiciones y la noción de "demostrable" pierde todo sentido.

2. **Completitud:** Dada una proposición **A**, como o bien **A** es verdadera o $\neg$**A** es verdadera, uno debería ser capaz de obtener o bien una demostración de **A**, o bien una demostración de $\neg$**A**.

Es decir, el sistema propuesto por Hilbert debía ser capaz de producir sólo verdades (nada más que la verdad) y todas las verdades (toda la verdad).

De este modo, la noción de verdad matemática se transformaría en algo objetivo: una proposición resultaría verdadera si y sólo si pudiera demostrarse.

Parecen requisitos razonables y las más grandes mentes matemáticas de la época creían que este proyecto era realizable. Observemos que hay algunos fragmentos de la matemática en donde estos dos requisitos efectivamente se cumplen.

Ejemplos:

- **La teoría del orden de Q.**

Supongamos que queremos establecer una lista de propiedades del orden de los números racionales de tal modo que resulte una axiomatización completa, en el sentido de que para todo enunciado **A** sobre el orden de **Q** o bien **A** o bien $\neg$ **A** se obtengan como teoremas (equivalentemente, que todos los enunciados verdaderos en **Q** se deduzcan a partir de los axiomas).

Consideremos primero los enunciados que establecen que la relación $\leq$ es un orden total.

1) $\forall x\ (x \leq x)$

2) $\forall x\ \forall y\ ((x \leq y \wedge y \leq x) \rightarrow x = y)$

3) $\forall x\ \forall y\ \forall z\ ((x \leq y \wedge y \leq z) \rightarrow x \leq z)$

4) $\forall x\ \forall y\ (x \leq y \vee y \leq x)$

Es claro que esta primera lista es insuficiente porque, por ejemplo, el enunciado que expresa la densidad del orden no puede obtenerse como un teorema de 1)-4), ya que el orden de los enteros verifica 1)- 4) y no es denso.

Añadimos entonces como nuevo axioma la densidad del orden.

5) $\forall x\ \forall y\ ((x \leq y \wedge x \neq y) \rightarrow \exists z\ (x \leq z \wedge x \neq z) \wedge (z \leq y \wedge z \neq y))$

Pero esto todavía es insuficiente porque, por ejemplo, el hecho de que **Q** no tiene máximo ni mínimo no puede obtenerse como un teorema, dado que los racionales positivos con el 0 y los racionales negativos con el 0 verifican 1) – 5).

Agregamos entonces como nuevos axiomas los enunciados que expresan que el orden de **Q** no tiene máximo ni mínimo.

6) $\forall x\ \exists y\ (y \leq x \wedge y \neq x)$

7) $\forall x\ \exists y\ (x \leq y \wedge x \neq y)$

La teoría cuyos axiomas son 1) – 7) se llama *teoría del orden lineal denso sin extremos* (TOLD). Puede probarse (ver (3)) que es una teoría de primer orden completa. Esto significa que para todo enunciado **A** de *primer orden* en el lenguaje cuyos símbolos son: = y $\leq$, vale que o bien **A** es un teorema de TOLD o bien $\neg$ **A** es un teorema de TOLD.

(Recordar que las fórmulas de *primer orden* de un lenguaje L son las que se obtienen a partir de fórmulas *atómicas* iniciales por medio de los conectivos lógicos usuales $\neg$, $\wedge$, $\vee$, $\rightarrow$, y por los cuantificadores $\forall$, $\exists$ aplicados a variables individuales numeradas v_i, $i \in N$. Las fórmulas atómicas varían en cada lenguaje y dependen de los símbolos de relaciones y funciones considerados. En nuestro ejemplo, las fórmulas atómicas son $x \leq y$, $x = y$ donde x e y representan cualquier variable numerada. No está permitida en cambio la cuantificación sobre subconjuntos, o sobre símbolos relacionales y funcionales. En particular, la propiedad de que todo subconjunto acotado superiormente tiene supremo, que diferencia el orden de **Q** del orden de **R**, no es expresable en primer orden.)

Del hecho de que TOLD es una teoría completa resulta también que para cualquier enunciado **A**, **A** es un teorema de TOLD si y sólo si **A** se verifica en **Q** , (o en cualquier otro ejemplo particular de conjunto ordenado que satisfaga los axiomas de TOLD). Es decir, en las teorías completas es lícito "probar con un ejemplo". Si llamamos T(**Q**) al conjunto de enunciados de 1er. orden verdaderos en **Q**, tenemos que TOLD es un subconjunto finito de T(**Q**) a partir del cual pueden obtenerse sintácticamente, como teoremas, todos los demás enunciados de T(**Q**). Es decir, en este caso, lo verdadero coincide con lo demostrable.

Otros fragmentos de la matemática para los que pueden obtenerse axiomatizaciones completas son las teorías de 1er. orden correspondientes a:

- álgebras de Boole sin átomos,
- cuerpos algebraicamente cerrados de característica 0,
- cuerpos real-cerrados,
- grupos abelianos divisibles sin torsión.

Para las correspondientes axiomatizaciones ver, por ejemplo, (3).

En el caso de las álgebras de Boole sin átomos la axiomatización es finita, como en TOLD. En los otros ejemplos, sin embargo, se requiere un conjunto infinito de axiomas. En la teoría de los cuerpos algebraicamente cerrados de característica 0, por ejemplo, el lenguaje es L= {+, ., =, 0, 1} y los axiomas son los usuales para cuerpos, (una cantidad finita), a los que se añade, para cada *n*, el axioma

$$\neg (\underbrace{1+1+\cdots 1}_{n} = 0)$$

y, para cada polinomio *p(x)* de coeficientes enteros, el axioma

$$(\exists x)\, p(x)=0$$

El hecho de que algunas axiomatizaciones resultan infinitas muestra la necesidad de precisar la noción de *teoría axiomática* . En efecto, dado cualquier ejemplo **M** de una teoría T con lenguaje L, el conjunto T(**M**) de todos los enunciados de L que son verdaderos en A proporciona siempre una teoría completa que extiende a T. Por lo tanto, este conjunto podría proponerse siempre en principio como una axiomatización completa. En este punto debe recordarse la intención original detrás de las axiomatizaciones, que es la de dar métodos mecánicos de demostración. En ese sentido, lo que se busca al proponer un conjunto de axiomas, es que, aunque sea infinito, resulte verificable sintácticamente por una computadora (es decir, que sea lo que se llama un conjunto *recursivo*, ver (7). Esto significa que, dado un enunciado cualquiera, una computadora pueda decidir en una cantidad finita de pasos, por una comprobación puramente sintáctica de símbolos, si el enunciado dado es o no un axioma (obsérvese que esto puede hacerse para la axiomatización propuesta para los cuerpos algebraicamente cerrados). En cambio, para verificar si un enunciado dado pertenece o no al conjunto T(**M**), la computadora debería ser capaz de decidir mecánicamente la verdad o falsedad en **M** de todos los enunciados, lo que intuitivamente parece mucho más difícil y, en general, como veremos, es imposible.

Diremos entonces que una teoría es *axiomática* si una computadora puede decidir, en una cantidad finita de pasos, si un enunciado dado es o no un axioma. La importancia de esta definición está dada por la siguiente proposición.

Proposición: *Si una teoría es axiomática, entonces una computadora puede generar mecánicamente todos los teoremas de la teoría.*

Demostración: Recordemos que una demostración es una lista finita de enunciados, cada uno de los cuales es un axioma lógico o un axioma de la teoría o se deduce de enunciados anteriores de la lista por las reglas lógicas de inferencia. Un teorema no es más que el último enunciado de la lista.

Podemos pensar entonces que todas las demostraciones están dispuestas en una matriz infinita A(i,j) donde la primera fila corresponde a las demostraciones de longitud 1 (axiomas) la segunda fila a las demostraciones de longitud 2, etc. Basta entonces programar a la computadora para que recorra esta matriz de acuerdo con el método diagonal de Cantor: A(1,1), A(1,2), A(2,1), A(1,3), A(2,3), A(3,1), A(1,4),...
De este modo, en un tiempo infinito la computadora genera todas las demostraciones posibles.■

Observemos que en general no será posible decidir en una cantidad finita de pasos si un enunciado es o no un teorema. En efecto: si el enunciado es un teorema, entonces aparecerá al cabo de un tiempo finito como última línea de alguna de las demostraciones. Pero si el enunciado no aparece con el correr del tiempo no hay modo de saber si definitivamente no aparecerá o es sólo que debe esperarse un tiempo más prolongado

.

Esto nos lleva a considerar un problema conexo al de la completitud que es el *problema de la decisión.*

- **El problema de la decisión**

Dado un sistema formal F el problema de la decisión para F es dar un algoritmo que permita decidir si un enunciado dado es un teorema o no. El sistema se dice *decidible* si tal algoritmo existe.

Observar, por ejemplo, que el cálculo proposicional es decidible: el algoritmo de decisión son las tablas de verdad.

Volviendo al sistema formal que propugnaba Hilbert, y que debía incluir a toda la matemática, es claro que pedir a este sistema un algoritmo de decisión parece un requisito demasiado fuerte: si hubiera tal procedimiento todas las preguntas matemáticas podrían ser contestadas mecánicamente. Sin embargo, si un sistema axiomático es consistente y completo (los dos requisitos que sí pedía Hilbert) entonces *tiene un algoritmo de decisión.* Para probar esto basta observar que si F es completo, y se nos da un enunciado **A**, o bien **A** o bien $\neg$ **A** es un teorema de F. Se puede entonces generar mecánicamente por el método de Cantor todas las demostraciones de F buscando en la última línea de cada una de ellas uno de estos dos enunciados. Al cabo de un tiempo finito puede contestarse si **A** es un teorema o no.

El programa de Hilbert era irrealizable como lo demostraron Gödel (1931), Rosser (1936) y Turing y Church (1936).

Más aún, no sólo es imposible dar una axiomatización completa de toda la matemática sino que tampoco puede darse una axiomatización completa ni siquiera de la aritmética elemental, esto es de los enunciados de primer orden verdaderos en los números naturales.

Para poder explicar estos resultados, recordemos primero la axiomatización original de la aritmética debida a Dedekind (1901), pero conocida como los axiomas de Peano.

Axiomas de Peano

1) $0 \in N$
2) $\forall x \in N \exists x' \in N$, *llamado el sucesor de x*
3) $\forall x \in N \neg(x'=0)$
4) $\forall x \ \forall y \ (x'=y' \rightarrow x=y)$
5) *Principio de inducción*
   Dada una propiedad P, si P(0) y $P(x) \rightarrow P(x')$, *entonces* $\forall x \ P(x)$.

Nótese que el principio de inducción 5) es equivalente a:

5') Si $S \subseteq N$ y
   (a) $0 \in S$
   (b) $x \in S \rightarrow x' \in S$
   entonces $S=N$.

También es equivalente al principio de buena ordenación:
5'') Si $S \subseteq N$ y $S \neq \varnothing$ entonces S tiene un elemento mínimo.

Este conjunto de axiomas no es, sin embargo, una teoría axiomática, en primer lugar porque utiliza nociones imprecisas como la de propiedad y presupone algo de la teoría de conjuntos, pero también porque el esquema 5') involucra a todos los subconjuntos posibles de N, es decir, a una cantidad no numerable de axiomas. Esto excedería la posibilidad, para una computadora, de generar todos los teoremas. Tampoco es posible expresar 5') o 5'') en primer orden porque necesitaríamos cuantificar sobre subconjuntos de N. Pero los axiomas de Peano sí son una teoría completa: puede probarse que dos estructuras cualesquiera N_1 y N_2 que verifican 1) – 5) son isomorfas (ver (6)). Por lo tanto, esencialmente hay un solo modelo. Entonces, un enunciado es un teorema si y sólo si es verdadero en este modelo.

Cuando uno se restringe a teorías axiomáticas para la aritmética debe utilizar enunciados de primer orden y la noción de subconjunto debe reemplazarse por la de *subconjunto expresable*, es decir por aquellos subconjuntos que están definidos mediante fórmulas, como los pares, los primos, los múltiplos de 7, etc. Nótese que, como el lenguaje es numerable, el conjunto de todas las fórmulas también es numerable. En definitiva, habrá sólo una cantidad numerable de conjuntos expresables.

Consideremos, por ejemplo, el sistema propuesto en el clásico libro de Mendelson (7).

Aritmética de Peano de primer orden

El lenguaje es L ={ ',+, ., 0}; los axiomas son:

$$S_1) \ x_1 = x_2 \rightarrow (x_1 = x_3 \rightarrow x_2 = x_3)$$
$$S_2) \ x_1 = x_2 \rightarrow x_1' = x_2'$$
$$S_3) \ 0 \neq x_1'$$
$$S_4) \ x_1' = x_2' \rightarrow x_1 = x_2$$
$$S_5) \ x_1 + 0 = x_1$$
$$S_6) \ x_1 + x_2' \rightarrow (x_1 + x_2)'$$
$$S_7) \ x_1 \ 0 = 0$$

S_8) $x_1\, x_2' = (x_1\, x_2) + x_1$

S_9) Para cada fórmula $A(x)$ de primer orden de L, el axioma

$$(A(0) \wedge (\forall x\, (A(x) \rightarrow A(x'))) \rightarrow \forall x\, A(x)$$

Observar que los números naturales pueden nombrarse en el lenguaje del siguiente modo: $0=0$, $1=0'$, $2=(0')'$, ...

El principio de inducción está dado por el axioma esquema S_9, pero como hemos observado, se limita sólo a los conjuntos expresables por fórmulas $A(x)$ del lenguaje L. A diferencia de la axiomatización original de Peano, pueden encontrarse modelos no isomorfos de esta teoría, por ejemplo, un modelo llamado *no standard* que tiene elementos "infinitamente grandes" ($x_0>0$, $x_0>1=0'$, $x_0>2=(0')'$, ...) .

K. Gödel en 1931, (completado por Rosser en 1936) probó el siguiente resultado fundamental sobre la limitación de los formalismos.

Teorema de incompletitud de Gödel: *Para cualquier teoría axiomática* T *que se proponga formalizar la aritmética elemental, si* T *es consistente, entonces* T *es incompleta.*

Por supuesto, esto vale para la aritmética de Peano de primer orden tal como figura más arriba, pero también para todas las extensiones axiomáticas que puedan proponerse por añadido de nuevos axiomas. Uno podría pensar que esta incompletitud esencial proviene de haber debilitado demasiado el principio de inducción. Sin embargo, hay una axiomatización *finita* debida a Robinson que también es incompleta y no admite extensiones axiomáticas completas. El sistema *RR* de Robinson axiomatiza el algoritmo de la división en vez del principio de inducción: es el algoritmo de la división, sumado al teorema de factorización en primos de la aritmética, lo que permite desarrollar el argumento principal de Gödel. Para una discusión completa sobre el teorema de Gödel y el sistema de Robinson ver (7).

La demostración original de Gödel puede verse como una formalización rigurosa en el lenguaje de la aritmética de la "paradoja del mentiroso", conocida también como la paradoja de Epiménides y que Smullyan reajusta en la siguiente versión:

Los habitantes de un país X reconocen a los del país Y porque o bien siempre mienten o bien siempre dicen la verdad. Con qué afirmación podría un recién llegado a X convencer a los oficiales de inmigración que no es un habitante de Y?

La afirmación es: "Yo miento".

Lo que hace Gödel es dar una codificación de las fórmulas de L y probar que "ser demostrable" es una propiedad expresable por una fórmula de L. Así, puede construir un enunciado que dice de sí mismo "Yo no soy demostrable". Pero entonces, este enunciado es verdadero si y sólo si no es demostrable. Esto da lugar a dos opciones: o bien el enunciado es verdadero y no demostrable, o bien es demostrable y falso. Pero, si se asume la consistencia del sistema, lo demostrable siempre es verdadero. Por lo tanto, en realidad, el enunciado construído por Gödel *es* verdadero, pero no puede demostrarse en el sistema axiomático.

Esta situación no puede ser reparada agregando este enunciado como nuevo axioma, porque en el nuevo sistema axiomático encontraríamos otro enunciado verdadero y no demostrable. Tampoco por el agregado de un conjunto recursivo

cualquiera de nuevos enunciados. Es decir, en tanto que se quiera mantener control sobre el conjunto de axiomas de manera que la noción de "demostrable" se corresponda con operaciones puramente sintácticas que puedan ser corroboradas mecánicamente por una computadora, la incompletitud es inevitable: habrá enunciados sobre los números naturales que son verdaderos, pero que no aparecerán en la lista de teoremas que genera la computadora.

Volviendo al enunciado construído por Gödel, observamos que todo sistema axiomático consistente da lugar a un plus de verdad que puede ser reconocido desde fuera del sistema (por consideraciones metamatemáticas), pero que es inaccesible para el propio sistema. Esto puede pensarse como una imposibilidad intrínseca de los sistemas axiomáticos (y de las computadoras) para reflexionar sobre sí mismos y sacar conclusiones sobre los alcances de su funcionamiento (autoreferencia). Este es uno de los principales argumentos de R. Penrose en su libro "The emperor's new mind", (Vintage, 1990), en contra de la posibilidad de que pueda modelarse el pensamiento humano con un algoritmo simulable por una computadora.

Los teoremas de incompletitud de Gödel y Rosser dejaban sin embargo abierta todavía la cuestión de la decisión. Esto es lo que terminaron de precisar, por separado, Turing (9) y Church (4) en 1936, dándole el golpe de gracia al programa de Hilbert: en su trabajo (9) *Sobre los números reales computables, con una aplicación al problema de la decisión*, Turing muestra que no es posible decidir en general y a priori si una computadora para un valor dado de entrada se detendrá o no. Esto es lo que se llama la indecidibilidad del *Halting Problem*. Es decir, hay un fragmento de la matemática que no puede axiomatizarse en el sentido que pedía Hilbert, ya que, como hemos visto, una axiomatización completa daría lugar a un método de decisión. Más aún, en ese mismo trabajo Turing muestra que todos los enunciados sobre el funcionamiento de sus máquinas de computar (llamadas luego máquinas de Turing), pueden expresarse en el cálculo de predicados de primer orden. Es decir, en particular demuestra que *el cálculo de predicados* es indecidible. A este mismo resultado había llegado Church de forma casi simultánea a través de la noción de función *efectivamente calculable* (que resulta equivalente a la de función computable por máquinas de Turing). Church logra además definir la noción de "efectivamente calculable" en el lenguaje que hemos considerado para la aritmética, lo que le permite obtener como un corolario la indecidibilidad de la aritmética.

El argumento de Turing es muy similar al argumento diagonal de Cantor para probar que el intervalo real (0,1) no es numerable. Se definen primeramente los números reales *computables*. Digamos que un número real es computable si hay un programa (en un lenguaje de computación fijado) que al ejecutarse va dando la lista de sus dígitos uno por uno. Esta lista de dígitos, agregando si es necesario una cola de ceros, puede siempre suponerse que es infinita. Ejemplos de números computables son π, e, y las raíces reales de polinomios de coeficientes enteros.

Como los lenguajes de computación tienen a lo sumo una cantidad finita de símbolos y los programas no son más que una lista finita de instrucciones escritas en este lenguaje, resulta que hay a lo sumo una cantidad numerable de programas (partes finitas del conjunto de instrucciones, que es numerable). Entonces hay una cantidad a lo sumo numerable de números reales computables. En particular, hay a lo sumo una cantidad numerable de números reales computables en el intervalo (0,1). Hacemos entonces una lista de todos los posibles programas, que incluirá a todos los números reales computables en (0,1), y escribimos junto a cada programa o bien la sucesión (infinita) de dígitos del número real que computa, si computa un número real entre

(0,1), o bien un blanco en cualquier otra opción, (si no computa un número real entre (0,1), o se detiene en algún dígito, o entra en un ciclo, o da cualquier señal de error).

Tenemos así un diagrama del tipo

P_1:

P_2: 0,1010010001.....

P_3: 0,333333.....

P_4:

P_5: π-3 = 0,141592.....

⋮

Es claro que todos los números reales computables en (0,1) por programas del lenguaje considerado están en alguna de estas filas. Lo que hacemos a continuación es reproducir el argumento diagonal de Cantor. Es decir, definimos un nuevo número real, entre 0 y 1, distinto a todos los listados, modificando los dígitos de la diagonal. Más precisamente, llamemos $b = 0, b_1b_2... b_n...$ a este número. Buscamos en el diagrama el primer programa de la lista que computa un número real entre 0 y 1; obtenemos b_1 eligiendo un dígito distinto al primer dígito que computa este programa. En nuestro diagrama sería $b_1 \neq 1$. Vamos al segundo programa de la lista que computa un número real entre 0 y 1. Obtenemos b_2 eligiendo un dígito distinto al segundo dígito que computa este programa. En nuestro diagrama, $b_2 \neq 3$. Etc.

De esta manera, hemos construído un número real b entre 0 y 1 distinto a todos los listados en el diagrama. Por lo tanto, b es no computable. Ahora bien, ¿por qué b es no computable, si prácticamente hemos indicado un método para obtener sus dígitos? Lo que ocurre es que para computar los dígitos de b necesitamos poder decidir si el programa siguiente en la lista al que da el dígito n para b va a arrojar o no el dígito n+1. En otras palabras, ¿qué ocurre si al ejecutarse este programa no aparece el dígito n+1? Esto podría deberse a que el programa entró en un ciclo y nunca lo arrojará, o bien a que no se ha esperado el tiempo suficiente para que ese dígito sea procesado. Esta es la pregunta que a priori no puede responderse. Si pudiera decidirse sobre esta cuestión, en el primer caso, en el diagrama figuraría un blanco y simplemente saltaríamos la línea al programa siguiente. En el segundo caso, obtendríamos el dígito n+1 de b de acuerdo con lo prescripto. Es decir, ¡b sería computable!

Este es el argumento original de Turing que le permite afirmar que no se puede decidir si un programa dado va a computar o no el dígito n-ésimo de un número real. Turing sabía que este mismo argumento valía en situaciones más generales y que se refería, en el fondo, a la imposibilidad de decidir si un programa cualquiera se detendrá o no para un valor dado de entrada. Al mismo tiempo, observa en su trabajo que si bien el argumento sobre los números reales es perfectamente legítimo deja la sensación de que "debe haber algún error". Es en el intento de precisar este argumento que define la noción que se conoce ahora como *máquina de Turing* , a partir de la que prueba luego la indecidibilidad del Halting Problem. Lo que haremos nosotros es precisar la noción de

programa para obtener este mismo resultado siguiendo los lineamientos de M. Davis en (5). Este será el tema de la segunda parte.

PARTE 2: Programas y codificación de Gödel

Desarrollaremos un lenguaje de programación lo más sencillo posible: tendremos una cantidad ilimitada de *variables de entrada*, que denotamos $X_1, X_2, \ldots$, una *variable de salida*, que denotamos Y, y *variables locales*, que denotamos $Z_1, Z_2, \ldots$ Tendremos, además, un conjunto $A_1, B_1, C_1, D_1, E_1, A_2, \ldots$ de *etiquetas*, que se colocarán (o no) a la izquierda de las instrucciones. Las variables toman valores en los números naturales.

Sólo consideraremos las siguientes cuatro instrucciones:

1) $V \leftarrow V+1$ (Sumar 1 al número natural que figura en la variable V.)

2) $V \leftarrow V-1$ (Restar 1 al número que figura en V, si este número es distinto de cero. En caso contrario, dejarlo invariante.)

3) $V \leftarrow V$ (Continuar, dejando invariante el valor de V.)

4) IF V≠0 GO TO L (Si el valor de Ves distinto de cero proceder a ejecutar la primera instrucción etiquetada con L. En caso contrario, continuar con la próxima instrucción. Notar que en ambos casos, el valor de V queda invariante).

Un *programa* es una lista finita de instrucciones, que pueden estar o no etiquetadas. Diremos que un programa *para* si una instrucción con etiqueta L está por ser ejecutada pero no hay instrucción asociada a esa etiqueta (en general usaremos la etiqueta $E = E_1$ por *Exit)*. Dado un programa **P**, convendremos que todas las variables locales, la variable Y, y todas las variables de entrada que no tienen valor asignado tienen inicialmente el valor 0.

Ejemplo 1

Consideremos el siguiente programa.

[A] IF $X \neq 0$ GO TO B
$Z \leftarrow Z+1$
IF $Z \neq 0$ GO TO E

[B] $X \leftarrow X-1$
$Y \leftarrow Y+1$
$Z \leftarrow Z+1$
IF $Z \neq 0$ GO TO A

Obsérvese que cualquiera sea el valor x de entrada que se asigne a X, el valor final de Y después de ejecutado el programa es x. Es decir, se puede pensar que este programa computa la función de una variable $f(x)=x$. Pero observemos que un programa **P** da lugar a una función $\psi_P^{(n)} : N^n \rightarrow N$, *cualquiera* sea n. En efecto, si n es mayor que la cantidad k de variables de entrada con valor especificado que ocurren en el programa, le asignamos a las variables de entrada sobrantes el valor 0.

Por otro lado, si n es menor que k, los valores extra de input son ignorados.

Esto nos conduce a la siguiente definición de *función computable.*

Definición: Una función $f: N^n \to N$ se dice *computable* si existe algún programa **P** tal que la función de n variables que computa **P** coincide con f, es decir:

$\psi_P^{(n)}(x_1,\cdots,x_n)$ está definida si y sólo si $f(x_1, ..., x_n)$ está definida y en tal caso $\psi_P^{(n)}(x_1,\cdots,x_n) = f(x_1, ..., x_n)$.

Ejemplo 2

Consideremos el siguiente programa:

$[A]$ $X \leftarrow X+1$
IF $X \neq 0$ GO TO A

Notemos que la función de una variable que computa este programa no está definida en ningún punto, porque para todo valor de entrada el programa entra en un ciclo infinito. Es decir, la función que no está definida en ningún punto *es* computable.

Ejemplo 3

Consideremos el siguiente programa, que llamamos el *macro* $V \leftarrow 0$.

$[L]$ $V \leftarrow V-1$
IF $V \neq 0$ GO TO L

Obviamente, el efecto de este programa es asignarle el valor 0 a la variable V.

Ejemplo 4

Nos proponemos dar ahora un programa que asigne a Y el valor de X pero que conserve en X el valor original (lo que no ocurría en el ejemplo 1).

[A] IF $X \neq 0$ GO TO B
GO TO C

[B] $X \leftarrow X-1$
$Y \leftarrow Y+1$
$Z \leftarrow Z+1$
GO TO A

[C] IF $Z \neq 0$ GO TO D
GO TO E

[D] $Z \leftarrow Z-1$
$X \leftarrow X+1$
GO TO C

En [B] se copia el valor de X en Y y en Z. En [D] se devuelve a X el valor original. Al finalizar el programa X e Y tienen ambos el valor original de X y Z tiene el valor 0. Llamamos a este programa el macro $Y \leftarrow X$.

Observemos sin embargo que este programa funciona bajo la condición de que Y y Z tienen valor inicial 0 (de acuerdo con nuestra convención). Si queremos generalizarlo para conseguir el programa que corresponda a $V \leftarrow V'$, donde V es una variable cualquiera, debemos tener la precaución de anteponer el macro $V \leftarrow 0$.
Es decir, la instrucción $V \leftarrow V'$ se obtiene escribiendo el macro $V \leftarrow 0$ y a continuación el macro $Y \leftarrow X$, reemplazando cada ocurrencia de la variable Y por V y cada ocurrencia de la variable X por V'.

Ejemplo 5

La función $f(x_1,x_2) = \begin{cases} x_1 - x_2, & si\ x_1 \geq x_2 \\ \text{no definida}, & si\ x_1 < x_2 \end{cases}$ puede computarse con el programa:

$Y \leftarrow X_1$
$Z \leftarrow X_2$

[C] IF $Z \neq 0$ GO TO A
GO TO E

[A] IF $Y \neq 0$ GO TO B
GO TO A

[B] $Y \leftarrow Y - 1$
$Z \leftarrow Z - 1$
GO TO C

En general, las funciones computables serán, como ésta, funciones parciales; su dominio de definición corresponde a los valores de entrada para los cuales el programa para.

Codificación de Gödel y el programa universal:

Nos proponemos ahora asignarle a cada programa un número natural de forma tal que, conocido el número, pueda decodificarse completamente el programa. Esta identificación entre programas y números naturales permite la construcción de un programa *universal*, es decir un programa que, para cada número natural ingresado como valor de entrada, decodifica el programa correspondiente a ese número y le asigna a Y el valor de salida correspondiente a ese programa.
La definición precisa de este programa universal, que consta de 18 instrucciones, puede verse en (5).

Consideremos en primer lugar la función $\langle x,y\rangle : N \times N \rightarrow N$ tal que
$\langle x,y\rangle = 2^x(2y+1)-1$.
Esta función es biyectiva. En efecto, es fácil ver, usando la unicidad de la factorización en números primos, que es inyectiva. Por otro lado, dado n, el par (x,y)

debe elegirse de modo que x es la mayor potencia de 2 que divide a $n+1$ e y se obtiene despejando en la ecuación: $2y+1 = (n+1)/2^x$. Notar que este proceso es computable.

Esto permite también definir una función biyectiva y computable de $N \times N \times N \to N$ dada por $<x,y,z> = <x,<y,z>>$.

Ahora disponemos las variables del lenguaje en la siguiente lista:

$$Y, X_1, Z_1, X_2, Z_2, X_3, Z_3, \ldots$$

Numeramos por separado también las etiquetas: $A_1, B_1, C_1, D_1, E_1, A_2, \ldots$

Escribimos $\#(V)$, $\#(L)$ para la posición correspondiente a la variable V y a la etiqueta L en estas listas. Por ejemplo, $\#(Y)=1$ y $\#(E)=5$.

Sea ahora I una instrucción (etiquetada o no). Entonces, podemos codificar a Y con el número $\#I = <a,<b,c>>$ donde a indicará si la instrucción está o no etiquetada, b indicará el tipo de instrucción, y c cuál es la variable que figura en la instrucción. Más precisamente:

1. Si I está etiquetada con etiqueta L, $a=\#L$; en caso contrario, $a=0$.
2. Si la variable que ocurre en I es V, $c=\#V-1$.
3. Si la instrucción en I es $V \leftarrow V$, $b=0$.
4. " " " " " " " " $V \leftarrow V+1$, $b=1$.
5. " " " " " " " " $V \leftarrow V-1$, $b=2$.
6. " " " " " " " " IF $V \neq 0$ GO TO L', $b=\#L'+2$.

Finalmente, sea **P** el programa $I_1, \ldots, I_k$. Entonces:

$$\#(\mathbf{P}) = (2^{\#I_1} 3^{\#I_2} \ldots p_k^{\#I_k}) - 1$$

Si ponemos como condición extra para los programas que la instrucción final no pueda ser $Y \leftarrow Y$ tenemos efectivamente una biyección computable entre programas y números naturales. En efecto, dado un número n, para decodificar el programa correspondiente basta hallar la factorización de $n+1$ en números primos (lo cual es un procedimiento computable). Los exponentes de estos factores primos se decodifican a su vez como instrucciones, que listadas en el orden creciente de los primos, dan el correspondiente programa.

Sea ahora $f(x_1, x_2, \ldots, x_n)$ una función computable por un programa **P** cuyas variables están entre $Y, X_1, \ldots, X_n, Z_1, \ldots, Z_k$ y cuyas etiquetas están entre $E, A_1, \ldots, A_t$. Asumimos que para cada instrucción "IF $V \neq 0$ GO TO A_i" hay efectivamente en **P** una instrucción etiquetada A_i (es decir, E es la única etiqueta de salida). Si no fuese así, siempre podemos añadir en cada A_i sin instrucción el macro GO TO E .

Escribimos:

$$\mathbf{P} = \mathbf{P}(Y, X_1, \ldots, X_n, Z_1, \ldots, Z_k, E, A_1, \ldots, A_t)$$

y notamos

$$\mathbf{P}_m = \mathbf{P}(Z_m, Z_{m+1}, Z_{m+n}, Z_{m+n+1}, \ldots, Z_{m+n+k}; E_m, A_{m+1}, \ldots, A_{m+t})$$

Queremos definir el macro $W \leftarrow f(V_1, V_2, \ldots V_n)$ con $W, V_1, V_2, \ldots V_n$ variables cualesquiera . Hacemos entonces:

$Z_m \leftarrow 0$
$Z_{m+1} \leftarrow V_1$
$Z_{m+2} \leftarrow V_2$
$\vdots$
$\vdots$
$Z_{m+n} \leftarrow V_n$
$Z_{m+n+1} \leftarrow 0$
$Z_{m+n+2} \leftarrow 0$
$\vdots$
$\vdots$
$Z_{m+n+k} \leftarrow 0$

$\mathbf{P}_m$
$[E_m]$ $W \leftarrow Z_m$

Observemos que m siempre puede elegirse lo suficientemente grande de modo que ninguna de las variables o etiquetas de $\mathbf{P}_m$ ocurra en el programa principal del cual este macro formará parte.

Finalmente, nos interesa obtener un macro para la instrucción:

IF $P(V_1, V_2, \ldots V_n)$ GO TO L

donde **P** es un *predicado computable*, es decir un predicado cuya función característica es computable.

El macro es:

$Z \leftarrow P(V_1, V_2, \ldots V_n)$
IF $Z \neq 0$ GO TO L

Con estos macros, estamos ahora en condiciones de dar una demostración precisa de la indecidibilidad del Halting Problem.

Halting Problem.

Consideremos el siguiente predicado binario:
Halt(x,y) $\Leftrightarrow$ el programa con número de Gödel y se detiene para el valor de entrada x.

Observar que si **P** es el programa tal que $y = \#\,\mathbf{P}$ y $\psi_P(x)$ es la función de una variable computada por **P**, entonces:

Halt(x,y) es verdadero si y sólo si $\psi_P(x)$ está definida.

Teorema: *Halt(x,y) es un predicado no computable.*

Demostración: Supongamos que *Halt(x,y)* fuese computable. Entonces podríamos considerar el siguiente programa:

P: [A] IF HALT(X,X) GO TO A

donde HALT(X,Y) es el programa que computa al predicado *Halt(x,y)*.

La función que computa este programa es:

$$\psi_P(x) = \left\{ \begin{array}{cc} \textit{indefinida} & \textit{si } Halt(X,X) \\ 0 & \textit{si } \neg Halt(X,X) \end{array} \right\}$$

Sea $y_0 = \#(P)$; entonces para todo x:

$Halt(x,y_0)$ es verdadero si y sólo si $\psi_P(x)$ está definida si y sólo si no es verdadero $Halt(x,x)$.

En particular, para $x=y_0$:

$Halt(y_0,y_0)$ es verdadero si y sólo si $Halt(y_0,y_0)$ no es verdadero. El absurdo proviene de suponer que el predicado $Halt(x,y)$ es computable.

Indecidibilidad del Halting Problem:

Corolario (asumiendo la tesis de Church)**:** *No hay un algoritmo que permita determinar para cualquier programa dado y cualquier valor de entrada si el programa parará o no con ese valor de entrada.*

Demostración: La tesis de Church nos dice que si existiera ese algoritmo, se correspondería con una función computable, es decir, que podríamos transcribirlo como un programa de nuestro lenguaje. Pero entonces, podríamos usar este programa para computar el predicado $Halt(x,y)$, absurdo.

PARTE 3: El número Ω de Chaitin.

Hasta ahora hemos visto que los programas pueden pensarse como números naturales. En particular, los consideraremos a partir de ahora en su expansión binaria, es decir, como sucesiones finitas de ceros y unos. La *longitud* de un programa **P** será la cantidad de bits de la sucesión y la denotaremos $|\mathbf{P}|$.

Consideremos ahora la siguiente sumatoria:

$$\sum_{P:P\ termina} 2^{-|P|}$$

donde **P** recorre los programas que paran con valor de entrada 0 (esto se corresponde con las máquinas de Turing iniciadas con la cinta vacía que paran.) Es fácil ver que cualquier programa **P** que pare con algún valor de entrada dado puede transformarse en uno de estos programas con valor de entrada 0.

En principio, si no se hacen restricciones sobre la forma de presentar los programas, esto es, si se admite que toda cadena de ceros y unos es un programa, la sumatoria escrita más arriba puede no converger. Para asegurar la convergencia de la serie, redefiniremos la noción de programa de modo que la extensión de un programa ya no pueda ser considerada un programa válido. Esto equivale en el fondo a expresar los programas en un lenguaje *sin prefijos*.

Lenguajes sin prefijos:
Un lenguaje L finito o infinito se dice *sin prefijos* si ninguna palabra de L es segmento inicial de otra o, equivalentemente, si ninguna palabra extiende a otra.

Ejemplo:

$$L=\{a^ib:\ i=0,1,2,..\}=\{b,\ ab,\ aab,\ aaab,...\}$$

La vinculación de los lenguajes sin prefijos con la convergencia de nuestra serie está dada por la siguiente:

Desigualdad de Kraft: *Sea $V_p=\{a_1,...,a_p\}$ un alfabeto con p letras . Sea L un lenguaje sin prefijos de alfabeto V_p. Entonces,*

$$0 \le \sum_{w\in L} p^{-|w|} \le 1$$

(para una demostración, ver(8))

Nuestro problema se reduce, entonces, a redefinir adecuadamente la noción de programa en un lenguaje sin prefijos. Esto se puede hacer anteponiéndole a cada programa que termina el dato de su longitud (también en notación binaria). Para poder diferenciar en una sucesión de ceros y unos dada cuál es el segmento que corresponde a la longitud y a partir de qué dígito empieza el programa se establece la convención de intercalar ceros entre los dígitos de la sucesión que corresponde a la longitud, añadiendo un uno al terminar. De este modo, el programa empieza cuando se detecta el primer uno en posición par.

Ejemplo:

Si el programa **P** es 110101, su longitud en sistema binario es $|\mathbf{P}| = 110$, y entonces se representará en el lenguaje sin prefijos como la sucesión: 101001110101.

Con estas precauciones podemos concluir:

Teorema (Chaitin): $\Omega = \sum_{P:P\ termina} 2^{-|P|}$ *es un número real entre 0 y 1.*

El número Ω de Chaitin tiene propiedades curiosas y desconcertantes.

La sucesión de sus dígitos es absolutamente aleatoria, y no podría distinguirse del resultado de una sucesión infinita de lanzamientos independientes de una moneda.

La información sobre los dígitos de Ω es lo que se llama una información *incompresible*. Esto significa que la manera más breve de dar los dígitos de Ω es la que resulta de escribirlos a todos uno por uno. Para aclarar quizá esto nótese la diferencia con el número de Champernowne: 0,12345678910111213... Se sabe que todos los dígitos del 0 al 9 ocurren en la expansión decimal de este número con la misma frecuencia 1/10. Este es uno de los requisitos que debe cumplir cualquier número aleatorio. Sin embargo, el número de Champernowne es, claramente, compresible. Dicho de otra manera, un jugador podría ganar infinitamente apostando a los dígitos de su expansión. En contraposición, la información sobre los dígitos del número de Chaitin no es obtenible por ningún algoritmo. Más aún, puede probarse que Ω codifica el Halting Problem en el siguiente sentido:

Teorema: *Dado i, si se conocen los primeros i bits de* Ω *puede decidirse el H.P. para cualquier programa de longitud* $\leq i$.

Demostración: Llamemos $0,b_1 b_2 \ldots b_i \ldots$ a la expansión binaria de

$$\Omega = \sum_{P:P\ termina} 2^{-|P|}$$

Notemos $\Omega_i = 0,b_1 b_2 \ldots b_i = \sum_{j=1}^{i} b_j 2^{-j}$

Por una observación anterior, basta mostrar que Ω codifica el Halting Problem para los programas **P** tales que el valor inicial de entrada es 0.

Notemos:

$$H = \{\mathbf{P}:\ \psi_P(0)\ \text{está definido}\}$$

Este conjunto puede ser numerado de forma computable por una función $g: N \rightarrow \{0,1\}^*$. En efecto, disponemos en una primera fila los programas que terminan después de ejecutada la primera instrucción, en la segunda fila, los que terminan después de ejecutada la segunda instrucción, etc. Tenemos así una matriz infinita pero la recorremos con el método diagonal de Cantor ya explicado.

Definimos, para $n \geq 1$, $W_n = \sum_{j=1}^{n} 2^{-|g(j)|}$

Es claro que la sucesión (W_n) es estrictamente creciente y converge a Ω.

Afirmación 1: Si $\Omega_i < W_n$, entonces $\Omega_i < W_n < \Omega \leq \Omega_i + 2^{-i}$.

En efecto,

$$\Omega = \sum_{j=1}^{\infty} b_j 2^{-j} = \sum_{j=1}^{i} b_j 2^{-j} + \sum_{j=i+1}^{\infty} b_j 2^{-j} \leq \Omega_i + 2^{-i},$$

pues $\sum_{j=1}^{i} b_j 2^{-j} = \Omega_i$ y $\sum_{j=i+1}^{\infty} b_j 2^{-j} \leq 2^{-i}$.

Supongamos ahora conocidos los primeros i bits de Ω. Esto equivale, por supuesto, a conocer Ω_i .Generamos *g(1), g(2)...*, hasta encontrar n tal que $\Omega_i < W_n$ (nótese que Ω_i da una condición para parar la búsqueda).

Sea ahora **P** un programa de longitud $i_1 \leq i$. Entonces:

Afirmación 2: **P** para si y sólo si **P** es *g(1)* o .... o **P** es *g(n)*.

⇐) Obvio.

⇒) Supongamos **P** = *g(m)* con $m>n$. Observar primero que:

$$W_m = \sum_{j=1}^{n} 2^{-|g(j)|} + \sum_{j=n+1}^{m} 2^{-|g(j)|} = W_n + \sum_{j=n+1}^{m} 2^{-|g(j)|}.$$

Como **P** = *g(m)* tiene longitud i_1 , $\sum_{j=n+1}^{m} 2^{-|g(j)|} \geq 2^{-i_1}$. En consecuencia:

$$\Omega > W_m \geq W_n + 2^{-i_1} \geq W_n + 2^{-i} > \Omega_i + 2^{-i} \geq \Omega$$

(La última desigualdad proviene de la afimación 1). Hemos llegado a un absurdo. Por lo tanto, si **P** termina y tiene longitud menor o igual que i, **P** es uno de los primeros n programas generados por g.

Bibliografía

1) X. Caicedo, *La paradoja de Berry revisitada, o la indefinibilidad de la definibilidad, y las limitaciones de los formalismos*, Lecturas matemáticas, Soc. Colombiana de Matemáticas, vol. XIV, (1993).

2) G. Chaitin, *The limits of mathematics,* Springer, 1990.

3) C.C. Chang, H. J. Keisler, *Model Theory,* Studies in logic, North Holland, 1990.

4) A. Church, *An unsolvable problem of elementary number theory*, American J. of Math., **58**, p. 345-363, 1936.

5) M. Davis, E. Weyuker, *Computability, Complexity and Languages*, Academic Press, 1983.

6) H.D. Ebbinghaus, J. Flum, W. Thomas, *Mathematical logic*, Springer, 1996.

7) E. Mendelson, *Introduction to Mathematical Logic*, (Fourth Edition), Chapman and Hall, 1997.

8) G. Rozenberg, A. Salomaa, *Cornerstones of Undecidibility*, Prentice Hall, 1994.

9) A. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, 2nd. Series, **42**, p. 230-265 (1936).