CONOCIMIENTO ABSOLUTO Y CONOCIMIENTO CIENTIFICO. UNA VISION COMPUTACIONAL

(Absolute Knowledge and Scientific Knowledge: A Computational View)

Alejandro SOBRINO*

Manuscrito recibido: 1999.3.22.
Versión final: 2000.5.29.

* Area de Lóxica e Filosofía da Ciencia, Universidade de Santiago de Compostela, Campus Universitario Sur, 15706 Santiago de Compostela. E-mail: lflgalex@usc.es

BIBLID [0495-4548 (2001) 16: 41; p. 269-299]

RESUMEN: El análisis de algunos programas lógicos y de algunos problemas ya tradicionales de la teoría de la computabilidad -como el problema de la correspondencia de Post-, permiten mostrar algunas claves para argumentar acerca de la posibilidad o imposibilidad de un ordenador omnisciente. Los programas lógicos inductivos y alguno de sus resultados más prometedores, como Golem, sirven para valorar la posibilidad de un ordenador como ayudante cualificado en la tarea de hacer ciencia. Ambas discusiones dan paso a una reflexión final sobre el mecanicismo.

Descriptores: indecidibilidad, automatización de la actividad científica, mecanicismo.

ABSTRACT: *The analysis of some logical programs and some traditional topics fromthe computability theory -as the Post Correspondence Problem-, enables us to illustrate some key points for discussing the possibility of a computer as an omniscient entity. Inductive logical programs and some of their most promising representatives, such as Golem, are showed useful for evaluating the possibility of a computer as a qualified support of scientific activity. Both topics give rise to a concluding discussion around mechanism.*

Keywords: *indecidibility, computational support of scientific activity, mechanism.*

SUMARIO

0. Preliminares: generalidades sobre la ciencia

La ciencia es una actividad humana especializada con la cual pretendemos conocer más y mejor acerca de nosotros mismos y de aquello que nos rodea. El conocimiento científico es un conocimiento especial que presupone la capacidad de usar un lenguaje no ambiguo y un método, el método científico, que garantiza -hablando un tanto idealmente- la objetivación de las aserciones, la corrección de las argumentaciones y lo fructífero de las predicciones.

Una de las características más señaladas de la actividad científica es su voluntad omniabarcadora; su propósito de explicar cada vez más fenómenos físicos y mentales, quizás hasta agotarlos todos. Los fenómenos físicos de la naturaleza, la estructura química de la materia o la composición biológica de los seres vivos, son algunas áreas que desde tiempo secular se han sometido al rigor del método científico. En cambio, ámbitos como los sentimientos, las intuiciones u otros 'fenómenos metales', han sido tradicionalmente considerados como no analizables o no sometibles a ese rigor. Considerados como aspectos típicamente humanos, han representado a menudo un reducto o bastión frente a lo inanimado (cosas) o lo biológicamente inferior (seres vivos sin conciencia), en cuyo estudio siempre se le han supuesto beneficios al método científico. Sin embargo, algunas entidades consideradas como mentales en el siglo pasado, son hoy explicables en términos biológicos o físicos. La polémica de hasta dónde progresará el programa de la Ciencia Unificada del neopositivismo; de qué parcelas psicológicas se adueñarán las Ciencias físicas y biológicas, sigue todavía viva.

Al mismo tiempo que la empresa científica expresa su voluntad expansionista, promoviendo conocer cada vez más cosas de terrenos hasta hace poco ajenos, el método científico asegura y reafirma la credibilidad de esta tarea al constituir un criterio de demarcación entre lo que debe considerarse ciencia y lo que se excluye por no serlo. Ciertamente, fuera del ámbito de la ciencia se reclaman saberes omniscientes pero, al no atenerse al método científico, no tienen para la ciencia ni credibilidad ni consideración. No obstante, y aún dentro del ámbito científico, cabe la hipótesis de que si todo pudiese ser objeto de conocimiento científico y se dispusiese de leyes científicas para todo, la entidad (humana, mecánica, etc.) que tuviese ese conocimiento sería omnisciente. Un ámbito propio de la ciencia dónde pueden hacerse y contestarse preguntas sobre la hipótesis de la existencia de una entidad omnisciente es el de la computabilidad.

Partamos de la ambiciosa conjetura de que todo, quizás, pudiese ser objeto de conocimiento científico o, seguramente de una forma más plausible, de creer que todo aquello valorable desde un punto de vista científico es susceptible, en mayor o menor grado, de cognoscibilidad científica. Si es así, el proceso de conocer estará asociado a la aplicación del método científico; dicho brevemente, a explicar fenómenos o ámbitos de conocimiento nuevo en términos de otros más conocidos. La explicación científica ha estado tradicionalmente asociada a establecer una relación normativizada, certificada o legalizada entre el explicatum o explanans, llamado también base de la explicación y el explicandum o explanandum, llamado también objeto de la explicación, de manera que se juzgue este último como una consecuencia inexcusable o muy fundamentada del primero. Si el explanans incluye leyes indubitablemente ciertas, la explicación es nomológico-deductiva; esto es, el explanandum se sigue necesariamente del explanans. Si la ley que une la base con el objeto de la explicación es de naturaleza probabilística, la explicación es probabilístico-inductiva y el explanadum se debe seguir con una probabilidad alta del explanans si es una explicación plausible del mismo. La certificación o legalización en la transmisión de la verdad la ofrece la lógica formal, que proporciona un lenguaje adecuado para hacer operaciones deductivas e inductivas; esto es, para calcular conclusiones que se siguen necesariamente o con una cierta credibilidad -una probabilidad, un porcentaje, p. ej.- de las premisas o información de partida.

El objetivo de este trabajo es analizar el tema de la posibilidad de conocimiento en el ámbito mecánico. En este contexto se entiende por máquina, 'Máquina de Turing' -a partir de ahora, MT-, que es la definición más universal de programa, aunque nos restringiremos en nuestro caso a los programas lógicos. Aunque un programa lógico es un caso especial de MT, las disquisiciones que se hagan en este trabajo sobre la posible omnisciencia de las máquinas no perderán generalidad por esto. Se parte del supuesto de que si alguna entidad fuese capaz de explicar científicamente todo, sería omnisciente. Por sus características en cuanto a capacidad y rapidez en la gestión de problemas, la computadora parecería un candidato ideal para ese atributo.

En este artículo se hace un examen de las posibilidades de una computadora para dar explicaciones desde una doble perspectiva: referidas al conocimiento exhaustivo de algún problema o al conocimiento de cualquier problema; en este último caso, se analizará propiamente la posibilidad de una computadora omnisciente. De este trabajo no debe esperarse

una tesis conclusiva, sino simplemente razones e ilustraciones para formarse una opinión fundada acerca de este tema. Con esta intención, se estructura así: en el apartado 1 se analizan algunos aspectos relacionados con los demostradores automáticos de teoremas, que explican -en el sentido ya aludido de 'explicación científica'- conocimiento nuevo a partir de datos conocidos. En el apartado 2 se muestra un sistema de deducción automático que encuentra el conocimiento verdadero que se puede deducir sobre un problema concreto. La simulación en Prolog de un caso típico, como el del mundo del mono y el plátano, muestra que, para problemas parciales bien definidos, hay entidades mecánicas que deducen correctamente cualquier verdad. Pero si se plantea la cuestión de si una computadora puede conocer todas las verdades, no de un ámbito concreto, sino de cualquier asunto, la respuesta es negativa. En el apartado 3 se analiza el problema de la correspondencia de Post, que muestra como hay ámbitos de conocimiento algorítmicos para los cuales una computadora no ofrece solución, a pesar de tenerla. En el apartado 4 se estudian las limitaciones que esto impone para calificar a una máquina como omnisciente. La falta de omnisciencia indica que ningún programa puede contener a priori la solución de cualquier problema, sino que, ante la aparición de un enigma concreto, el programa debe tratar de proveer su solución. En el apartado 4 se analizan aspectos relacionados con la creatividad y los programas. Se expone un programa Prolog para el aprendizaje del concepto de 'arco' y se ilustra con ello algunas ideas básicas de la programación lógica inductiva. En el apartado 5 se analiza la utilidad de los programas inductivos en la tarea de la explicación científica -al menos desde un punto de vista inductivista y falsacionista de la ciencia-, ya que ayudan a establecer hipótesis y ejemplos relevantes para su confirmación o falsación. Se exponen los trazos generales de *Golem*, programa de aprendizaje inductivo que descubrió una ley de la Bioquímica. Para terminar, se discute brevemente acerca de la posibilidad del mecanicismo y del papel que en esta discusión pueden tener los programas inductivos.

1. La mecanización de la explicación científica: la deducción automática de teoremas

Con la deducción automática se ha intentado mecanizar la tarea deductiva con el objeto de construir razonadores que, de modo artificial, expliquen o ayuden a explicar (en el caso de los sistemas de ayuda a la toma de deci-

siones) algunos ámbitos de conocimiento humano. Desde *Logic Theorist*, de A. Newell, B. Shaw y H. Simon, que demostró automáticamente algunos teoremas de los *Principia Mathematica*, hasta *Golem*, de S. Muggleton y C. Feng que, usando técnicas de programación lógica inductiva, descubrió una ley de la naturaleza en el campo de la estructura de las proteínas, ha habido intentos muy notables en esta dirección. Los patrones de explicación basados en lógica (los célebres *If...then*) han jugado un papel relevante en un área de investigación avanzada de la Informática que tiene por objeto simular conductas intelectuales de las que los humanos diríamos que son inteligentes. Se trata, como no, de la I. A., Inteligencia Artificial o Informática Avanzada. Y todos diríamos que suele ser una conducta inteligente de un humano dar explicación científica, en los términos señalados antes; esto es, deducir, para una situación o estado de cosas, el explicandum del explicatum.

Daremos a continuación una pincelada histórica del ámbito de la deducción automática. La demostración automática tiene sus orígenes en el propósito de G.W. Leibniz de disponer de una *lingua characteristica* que permita expresar, de una forma precisa, nuestros pensamientos, y de un *calculus raciocinator* que, al estilo de una operación matemática, posibilite resolver cualquier operación razonadora de la mente como el cálculo de la corrección de ese razonamiento. G. Boole concretó la idea de que las fórmulas algebraicas pudieran ser usadas para expresar relaciones lógicas y C.E. Shanonn halló una isomorfía entre las funciones de verdad booleanas y las funciones de conmutación de los circuitos, implementando operaciones booleanas como puertas lógicas. G. Frege extendió la *lingua* y el cálculo para razonamientos en los que se distinguen individuos y propiedades o relaciones como entidades lógicas notables. Por fin, Turing puso las bases teóricas de la maquinaria computadora que podría llevar a cabo mecánicamente -amén de otras- esas operaciones lógicas, apropiadamente dispuestas.

La deducción automática nace como un intento de limitar el vocabulario y las reglas de la lógica de Frege, acotando el poder generativo y transformatorio de la gramática de la *lingua*: por una parte, se admiten menos reglas de producción o de formación de fórmulas, de tal manera que se restringe el formato de 'oración bien formada' a menos casos; por otro, se limita el número de reglas deductivas a una única regla. Th. Skolem y J. Herbrand posibilitaron lo primero, al proponer las formas normales y estructuras Herbrand, demostrando -con el Teorema de Gödel-Herbrand-Skolem- que, dada una sentencia S en forma de Skolem, S es

satisfacible si y sólo si el conjunto de casos de la base Herbrand de S (abreviadamente, CBH(S)) es satisfacible. Si a este teorema se le une el teorema de compacidad es posible formular el siguiente enunciado: una sentencia S en forma de Skolem es insatisfacible si y sólo si existe un subconjunto finito de CBH(S) que es insatisfacible (en el sentido de la lógica proposicional). Este resultado, llamado Teorema de Herbrand, es la base de los procedimientos de decisión del cálculo de predicados; hablando en términos computacionales, de la existencia de algoritmos que se detienen tras un número finito de pasos en los casos en los que la cuestión se resuelva afirmativamente. J.A. Robinson, basándose en indagaciones hechas por W.V.O. Quine, D. Prawitz y M. Davis & H. Putnam sobre unificación de literales, propuso la regla de resolución como única regla deductiva.

Surge así el cálculo de resolución, que fundamenta la prueba automática de teoremas. Al disponer de un formato de fórmulas limitado y una única regla de inferencia, la confianza de implementar ese cálculo en un ordenador se acrecentó. El éxito llegó con las aportaciones teóricas de R. Kowalski en Edimburgo y la implementación de A. Colmerauer y Ph. Roussell en Marsella, creando el lenguaje de programación con lógica o fundamentado en lógica, al que denominaron PROLOG (PROgrammation in LOGique). Un intérprete o compilador de Prolog es un demostrador automático donde la pregunta que se le hace al sistema es tomada como un teorema conjetural. La respuesta, que en Prolog será booleana con posibles instanciaciones de variables en el caso afirmativo, constituirá una explicación -en el sentido aquí manejado- a la pregunta planteada. La traza de la respuesta mostrará, de manera exhaustiva, el proceso deductivo que ha llevado al intérprete a dar esa respuesta.

Examinaremos a continuación un ejemplo de cómo con un cálculo de resolución y, posteriormente, con su implementación en Prolog, se puede explicar una determinada conducta en términos de la deducción de una secuencia de acciones posibles tendente a conseguir un fin (la conclusión del problema). El ejemplo al que aludiremos, propuesto por McCarthy en el inicio de la I. A. simbólica, se le denomina el problema del mono y el plátano.

2. Modelización de un problema concreto: el mono y el plátano

Usualmente los humanos decimos de un animal que muestra cierta inteligencia si, siempre que no pueda conseguir un objetivo de forma directa, es

capaz de usar o idear situaciones intermedias que le acerquen a ese fin. El caso del mono y el plátano atiende a esta concepción y se plantea así: supongamos que hay una habitación en cuyo centro está colgado un plátano. El plátano está a una altura que lo convierte en inasible para el mono. El mono está situado en la puerta de entrada a la habitación, y la habitación tiene, en una parte alejada de la puerta de entrada, una ventana, a cuyo lado se sitúa una silla. El mono puede usar la silla para subirse a ella y obtener el plátano, pero antes ha de desplazar la silla al centro de la habitación y él mismo ha de trasladarse desde su posición original en la puerta hasta la ventana donde se encuentra la silla. La pregunta es: dado un estado inicial, que es el descrito, ¿puede emprender el mono una secuencia de acciones que le conduzcan a obtener el fin deseado; esto es, a conseguir el plátano? ¿Se puede simular mecánicamente el presumible comportamiento inteligente del mono con una máquina de la I. A. simbólica, como una máquina-Prolog? La respuesta es positiva y la vamos a documentar a continuación tanto en términos de un cálculo de resolución como de su análogo mecánico; esto es, de un programa Prolog.

En términos del cálculo de resolución, usamos un predicado P que actúa como un operador con cuatro ranuras que fijan la situación relativa del mono, la silla y el plátano. El propósito de la computación es encontrar una serie de aplicaciones de los "operadores" disponibles para transformar un estado inicial en un estado final deseado (donde el mono haya conseguido el plátano). Considérense las siguientes cláusulas, para las que, acto seguido, se dan las siguientes interpretaciones (entrecomilladas):

(Cl. 1) {P(a, b, c, d)}

"En la situación inicial *d*, el mono está en la posición *a*, el plátano está colgado en la posición *b*, y la silla está en la posición *c*". (a, b, c, d son constantes).

(Cl. 2) {¬P(x, y, z, s), P(w, y, z, *caminar*(x, w, s))}

"Si, en una situación s, el mono está en la posición x, entonces una aplicación de la función *caminar*(x, w, s) provoca que el mono se sitúe posteriormente en la posición w. En otras palabras, el mono está capacitado para caminar de una posición a otra". (x, y, z, s, w son variables).

(Cl. 3) {¬P(x, y, x, s), P(w, y, w, *empujar*(x, w, s))}

"Si el mono está en la misma posición x que la silla, entonces puede empujarla hacia cualquier posición w".

(Cl. 4) {¬P(x, y, x, s), P(x, y, x, *subir*(s))}

"Si el mono está en la misma posición que la silla, entonces puede subirse a la silla".

(Cl. 5) {¬P(x, x, x, *subir*(s)), *conseguir*(*coger*(*subir*(s)))}

"Si el mono está subido a la silla y si la posición del mono, la silla y el plátano coinciden, entonces el mono podrá conseguir el plátano cogiéndolo".

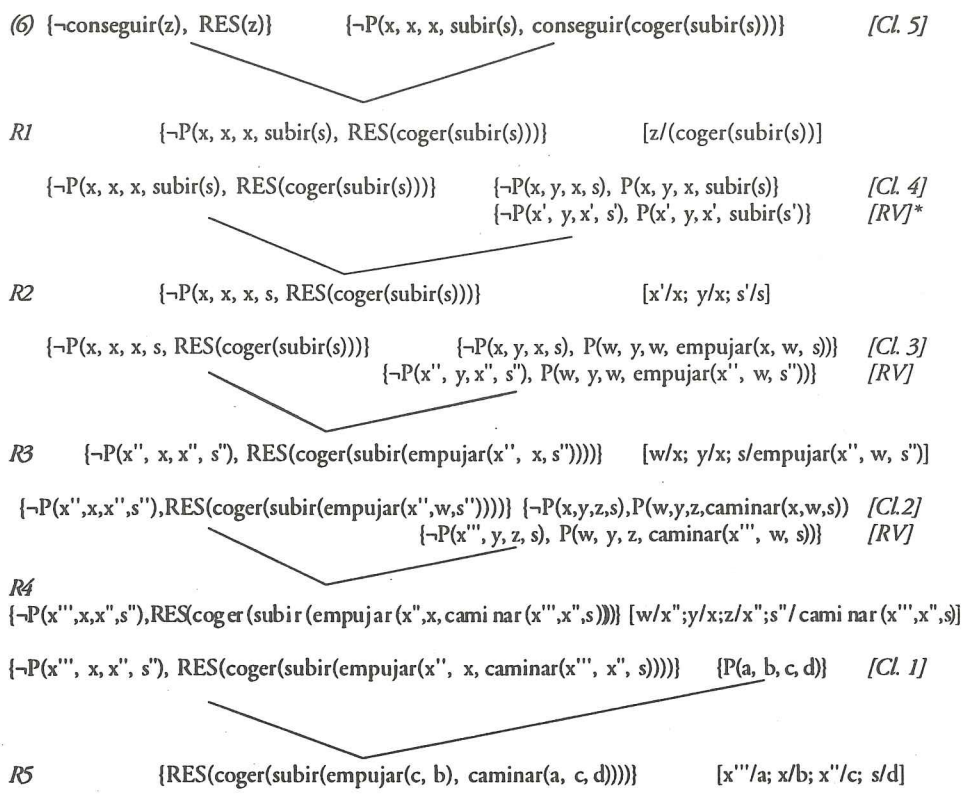
Las cláusulas describen el contexto del problema y representan la información declarativa. Sea ahora la pregunta:

$$\exists z \; \textit{conseguir}(z),$$

esto es, "¿hay alguna situación en la que el mono haya conseguido el plátano -y, si fuese así-, cómo lo logró?". Según el procedimiento refutativo del cálculo clausal, se niega la pregunta y se traduce a forma clausal, incluyendo un predicado de respuesta (RES) para que la sustitución de la variable z -si la hay- sea una solución explícita al problema.

(6) {¬*conseguir*(z), RES(z)}.

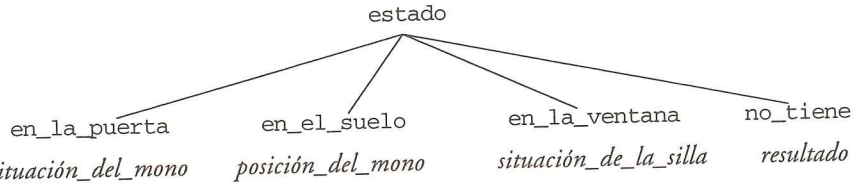
La siguiente secuencia R_1,..., R_5 proporciona una prueba de resolución de la cláusula-respuesta:



* Redenominación de variables

La interpretación de R_5, que representa la deducción final, es la siguiente: "Partiendo de la situación d, caminar desde a hasta c, empujar la silla desde c hasta b, subirse a la silla y coger el plátano".

Veamos ahora la formulación del problema en un lenguaje de programación lógica como Prolog (Cfr. Bratko, 1990[2]). Para representar el mundo posible del mono, utilizamos una estructura. Escojamos a la palabra *estado* como el functor que agrupa la información relativa a la situación posible del mono, la silla y el plátano. La siguiente figura muestra el estado inicial.



La resolución del problema se puede ver como un juego unipersonal con las siguientes reglas:

1. El objetivo es una situación en la que el mono tenga el plátano; esto es, cualquier estado en el cual el último componente es 'tiene' (el resto de las ranuras puede ser cualquier cosa):

estado(_,_,_,tiene).

2. Segunda: ¿cuáles son los movimientos que cambian el mundo del mono; esto es, que le permiten pasar de un estado a otro más interesante para lograr su fin? Hay cuatro tipos de movimientos:

(1) coger el plátano.
(2) subir a la silla. (Se asume que la acción de subir no es reversible).
(3) empujar la silla.
(4) desplazarse por la habitación.

Pero el mono no puede realizar cualquier movimiento independientemente del estado en el que esté. Por ejemplo, no puede desplazarse con la silla si no está en la misma posición que ella. Tales reglas se pueden formalizar en Prolog como una relación triádica llamada *movimiento*:

movimiento(Estado1, Movimiento, Estado2)

donde *Estado 1* es el estado antes del movimiento, *Movimiento* es el movimiento ejecutado y *Estado2* es el estado después del movimiento. El programa que se detalla a continuación da cuenta de los movimientos posibles:

```
mov(estado(en_el_centro, sobre_la_silla, en_el_centro, no_tiene),
   coge,
   estado(en_el_centro, sobre_la_silla, en_el_centro, tiene)).
mov(estado(P, en_el_suelo, P, H),
```

```
    sube,
    estado(P, sobre_la_silla, P, H)).
mov(estado(P1, en_el_suelo, P1, H),
    desplaza(P1, P2),
    estado(P2, en_el_suelo, P2, H)).
mov(estado(P0, en_el_suelo, P1, H),
    anda(P0, P1),
    estado(P1, en_el_suelo, P1, H)).
```

La cuestión principal a la que tiene que responder el programa es: ¿puede el mono en algún estado inicial *Estado* alcanzar el plátano? Se puede formular esta cuestión con un predicado de un argumento:

éxito(Estado)

donde *Estado* es el estado inicial del mono y *éxito* es el nombre de la relación que indica el logro del objetivo. El programa para *éxito* se basa en dos observaciones:

1. Para cualquier estado en el que el mono ya tenga el plátano, el predicado *exito* es verdadero; no se necesita ningún movimiento en ese caso. Esto corresponde al hecho:

```
exito(estado(_,_,_,tiene)).
```

2. En otras situaciones, se hace necesaria una definición recursiva que permita uno o más movimientos; esto es, el mono tiene éxito en cualquier estado *Estado1*, si hay algún movimiento *Movimiento* de *Estado1* a *Estado2* tal que el mono pueda coger el plátano en el *Estado2* (ejecutando para ello 0 o más movimientos).

```
exito(Estado1):-
    movimiento(Estado1, Movimiento, Estado2),
    exito(Estado2).
```

Con estas dos cláusulas se completa el programa del mono y el plátano. Algunas preguntas en las que es posible comprobar cómo el compilador deduciría automáticamente verdades útiles para la conducta de un mono serían las siguientes:

```
?- exito(estado(X, Y, en_el_centro, tiene)).
X = en_el_centro,
Y = sobre_la_silla;
```

```
No
?- exito(estado(en_la_puerta, en_el_suelo, en_la_ventana, no_tiene)).
Yes
```

Nótese que si bien la inteligencia que se le presupone a un mono es pequeña si se compara con la normal atribuible a un humano, las líneas de código necesarias para simular un acto concreto de esa inteligencia son realmente muy pocas.

Este ejemplo muestra que, para un sistema cerrado y completo -como el mundo del mono- los demostradores automáticos saben (pueden deducir) todo el conocimiento verdadero de ese mundo. Desde un punto de vista científico, por su capacidad de cálculo y de memoria, una máquina computadora parecería ser un candidato idóneo para reclamar el atributo de omnisciencia, pero siempre que fuese capaz de expresar cualquier verdad, no sólo las relativas al problema considerado. Las máquinas computadoras saben acerca de realidades algorítmicas; esto es, de ámbitos donde las cuestiones están definidas con un lenguaje preciso y existen procedimientos de decisión para, en un número finito y secuencial de pasos, proporcionar una explicación, -en este contexto, deducir la información requerida a partir de la información ya disponible-. La pregunta que se plantea a continuación es: ¿puede una máquina deducir información acerca de cualquier ámbito?; ¿puede ser global o totalmente cognoscente? La respuesta es negativa y resulta paradójico que haya sido A.M. Turing, el creador de las máquinas computadoras abstractas, el que haya establecido este resultado limitativo para las mismas. En lo que sigue trataremos de esclarecer algunos aspectos de este resultado.

3. Omnisciencia mecánica: el PCP

Como es conocido en el ámbito de la computabilidad, la tesis de A. Church asegura que, si un problema tiene solución algorítmica, entonces puede ser resuelto por una máquina de Turing (MT). Esto es, que los términos 'procedimiento efectivo' o 'computación algorítmica' quedan elucidados por una MT. La pregunta que se plantea a continuación es si hay una MT que, para un entrada genérica, agote el cómputo de esa entrada y termine con una respuesta. La contestación es negativa ya que hay algunos problemas que, a pesar de tener una definición algorítmica, no tienen solución; son insolubles. El ámbito de la matemática y los lenguajes formales proporciona algunos ejemplos al respecto.

Un número es par si es divisible por 2 y esto puede programarse en una computadora de tal forma que diagnostique, para una entrada numérica, si es o no par. Por tanto, una máquina teórica de computar que reciba como entrada a (N + programa), debería dar como resultado el conjunto y sólo el conjunto de los números pares. Pero, ¿se detendrá en algún momento la computadora al imprimir un nuevo número?; esto es, ¿hay algún número n tal que no hay otro mayor que él?. La respuesta es, desde luego, negativa, porque para cualquier n arbitrario, el número 2n es par y mayor que n. En conclusión, el dominio del problema es infinito, pero hay una respuesta para la pregunta y esta es negativa.

Ahora bien, para dominios infinitos, hay problemas que pueden ser formalizados en un cálculo de resolución de predicados que, aún estando perfectamente definidos, no ofrecen soluciones globales o de conjunto, sino sólo respuestas parciales. El problema de la correspondencia de Post (PCP) ilustra teóricamente, de modo clásico, un enigma resoluble que, a veces, carece de solución mecánica. El PCP se puede enunciar con una instancia y una pregunta:

Instancia: Una secuencia $(x_1, y_1),...,(x_k, y_k)$ de pares de cadenas no vacías sobre el alfabeto {0,1}.

Pregunta: ¿Existe una secuencia finita de índices $i_1, i_2,..., i_n \in \{1,..., k\}$, $n \geq 1$, tal que $x_{i1}, x_{i2},..., x_{in} = y_{i1}, y_{i2},..., y_{in}$?

Ya que estamos debatiendo si una máquina puede catalogarse como omnisciente, esta pregunta demanda una contestación en términos de cuál sería el comportamiento de una computadora que recibiese como entrada una ejemplificación cualquiera de la instancia del PCP enunciada arriba.

Lo que sigue es un ejemplo de PCP irresoluble. Sea i un conjunto de índices para x e y resumidos en la siguiente tabla:

i	x_i	y_i
1	10	101
2	011	11
3	101	011

Tratemos de encontrar una solución al PCP. El único emparejamiento posible es:

$x_1 = 10$
$y_1 = 101$

Las alternativas que se presentan a continuación son: i = 1 o i =3

Si i = 1
$x_1x_1 = 1010$
$y_1y_1 = 101101$

No emparejan. Por tanto, elegimos

i = 3
$x_1x_3 = 10101$
$y_1y_3 = 101011$

El problema queda de nuevo planteado con la alternativa i = 1 o 3, cuya solución (i = 3) dejaría las cosas igual que antes. Luego, el problema es irresoluble para una máquina, aunque un humano pueda 'ver' que, para esa tabla, no hay solución al PCP.

Lo anteriormente expuesto atañe a un ejemplo concreto, pero en general, para plantear el problema de la resolubilidad o irresolubilidad mecánica de un PCP cualquiera, deberíamos ser capaces de expresarlo en términos de una MT. En ese caso, el análogo computacional del problema planteado arriba sería mostrar que una MT que recibiese como entrada esa ejemplificación del PCP computaría soluciones paso a paso, pero nunca se detendría, aunque el problema, por ser algorítmico, tiene o no tiene solución de cada vez.

A fin de mostrar que la resolubilidad del PCP está inextricablemente vinculada a la aceptación por una MT de una cadena dada, se enuncia el problema de la correspondencia de Post modificado (PCPM) y se muestra que el PCPM tiene solución syss hay una MT que lo computa. En (Hopcroft & Ullman 1979) se puede consultar cuál es el formato de las subcadenas que corresponden a las instrucciones de una MT.

4. Limites a una computadora omnisciente

El PCP pone de manifiesto que, en general, no hay un formato uniforme de expresión de problemas que asegure su solución. Cada problema (i.e., demostrar una propiedad, un teorema,...) deberá ser formulado de una manera particular y su solución no podrá ser generada por un algoritmo o procedimiento general. Una máquina computadora que intente dar respuestas correctas a ciertos problemas relevantes, como el PCP, necesitará compu-

tar siempre y, en consecuencia, no podrá ofrecer una solución definida en un tiempo finito.

Anotamos a continuación dos consecuencias que se siguen de la carencia de un algoritmo para decidir si una MT se detiene o no para cualquier entrada -tesis de Turing-: una primera consecuencia es que no hay una máquina de propósito absolutamente general que trate cualquier problema y encuentre su solución. Por tanto, una mente científico-mecánica nunca podrá ser omnisciente. La ciencia pone coto a su propio conocimiento, porque mientras el dominio de problemas es infinito, las soluciones a los mismos se incrementan sólo en tiempo finito. En efecto, aunque en una MT la rapidez de cálculo pudiese ser tan elevada como se desease, siempre tendrá un valor finito. Como la cinta de la MT es infinita, siempre quedarán problemas sin tratar. Usando una metáfora, es algo análogo a qué lugares podemos visitar (acompañados de Einstein). La velocidad a la que podríamos viajar, aún teniendo un valor muy alto, tiene un límite (la velocidad de la luz); en cambio, las distancias son virtualmente ilimitadas. Por tanto, habrá lugares que nunca podrán ser visitados.

Otra consecuencia es que hay límites a la posibilidad de reconocer la realidad física. El propósito positivista de modelar lógicamente toda la realidad para explicarla queda en entredicho. Hay realidades que aún siendo verdaderas, no podrán ser tratadas formalmente como tales; por tanto, la lógica no podrá explicarlas. En consecuencia, o bien la inteligibilidad de la realidad como un todo no tiene causa o su causa está fuera del mundo racional. Una computadora no puede ser perfectamente racional. El trabajo creativo no puede ser reemplazado por alguna técnica, procedimiento o sistema descrito por la mente antes de ejecutarlo.

¿Favorecen estas conclusiones el postular que, de haber entidades omniscientes, son ajenas y están más allá de una mente mecánica o científica? O dicho de otra forma, ¿hay que buscar la omnisciencia fuera de la ciencia? En cierto modo, parecería seguirse una respuesta positiva según lo discutido hasta aquí. Como indicamos, una de las tareas del método científico es distinguir; separar lo que es ciencia de lo que no lo es. Y claramente, por lo que acabamos de decir, la omnisciencia no pertenece al dominio de la ciencia. Pero esta conclusión no deber dar pie a concluir, al menos desde una perspectiva científica, que se encuentra fuera de la ciencia. El método científico postula que la realidad es la única fuente de verdad, aunque no contenga la causa de 'la verdad' y que la realidad sólo se trata de modo interesante con su método. Lo demás, aplicando su criterio de demarcación, simplemente no pertenece a su discurso.

Decíamos antes que una consecuencia de la tesis de Turing es que no hay una máquina que sea absolutamente racional; esto es, que explique todo problema. La creatividad mecánica está asociada a las soluciones que proporciona la computadora y no a una descripción *a priori* de cualquiera de ellas. En el siguiente apartado expondremos un programa lógico que adquiere conocimiento a partir de la realidad que modela: es un programa de aprendizaje inductivo.

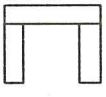
5. Creatividad y programas

Si bien, como se ha argüido, una computadora no puede ser omnisciente, cabe preguntarse si puede ayudarnos a tener cada vez más conocimiento científico de nosotros mismos y de lo que nos rodea. Esta cuestión ya quedó parcialmente contestada cuando indicamos que, para problemas concretos y bien definidos, la computadora es capaz de deducir cualquier conocimiento verdadero. Pero cabe preguntarse si las computadoras pueden ayudar no sólo a dar cuenta del conocimiento ya establecido, sino a descubrir nuevos hechos. La programación lógica inductiva es un área pujante en la que se han conseguido interesantes resultados. Ilustraremos el tema con dos ejemplos: *Arches*, programa lógico que muestra cómo una máquina aprende el concepto de 'arco' y *Golem*, que con una metodología similar a la de *Arches*, consiguió descubrir una ley de la Bioquímica.

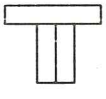
Arches es un programa Prolog desarrollado por Bratko -aunque retomado de Winston- que aprende conceptos relacionados con formas estructurales en un espacio bidimensional. El programa aprende el concepto de 'arco' a partir de un lote de ejemplos y contraejemplos que le proporciona el que le pretende enseñar. Los ejemplos son procesados secuencialmente y, con cada nuevo caso, el programa tiene que actualizar la hipótesis que se va configurando acerca de ese concepto. Sean estos cuatro casos ejemplos y contraejemplos del concepto de 'arco'.

Ejemplos positivos (+) y negativos (-) de 'Arco'

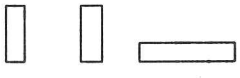
Ejemplo 1
Arco +



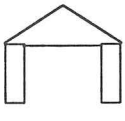
Ejemplo 2
Arco -



Ejemplo 3
Arco -



Ejemplo 4
Arco +



El proceso de aprendizaje se conforma estableciendo una hipótesis inicial tomada de un ejemplo positivo o prototípico de 'arco' y refinando esa hipótesis de comienzo con sucesivas incorporaciones de ejemplos. Una vez procesados los cuatro ejemplos y actualizadas las sucesivas hipótesis, la propuesta final debería mostrar lo siguiente respecto a las partes constitutivas del concepto y a las relaciones entre las partes:

Partes:
- Un arco tiene tres partes: llamémosle *columna1*, *columna2*, y *dintel*
- *columna1* y *columna2* son rectángulos; *dintel* puede ser una figura más genérica, un polígono, p. ej., (se sigue de los ejemplos 1 y 4)

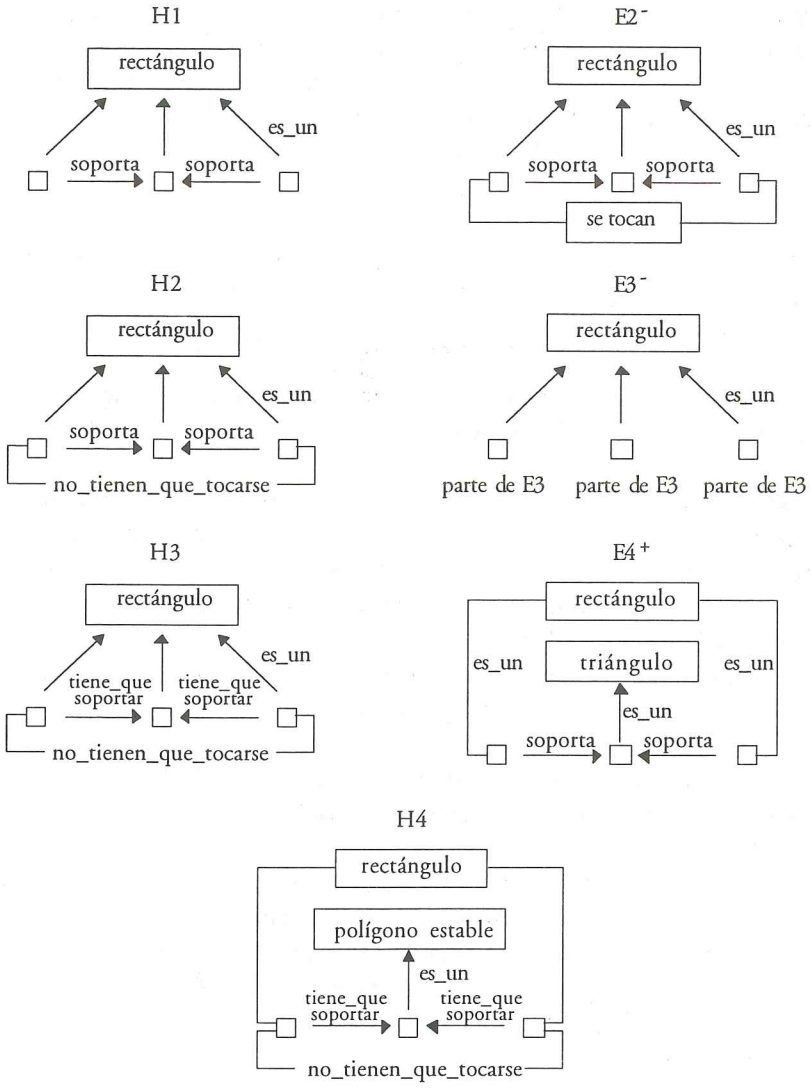
Relaciones:
- *columna1* y *columna2* no deben tocarse (se sigue del ejemplo 2)
- *columna1* y *columna2* deben soportar al *dintel* (se sigue del ejemplo negativo 3).

Cuando se aprende un concepto por procesamiento secuencial de ejemplos, el proceso de aprendizaje recorre una secuencia de hipótesis H1, H2, etc., cada una de las cuales representa una aproximación al concepto, ya que resulta de una actualización de la hipótesis anterior. Este proceso se hace explícito en el siguiente algoritmo:

Para aprender un concepto C dada una secuencia de ejemplos E_1, E_2,..., E_n (donde E_1 debe ser un ejemplo positivo de C),

```
Empezar
    Tomar a E1 como la hipótesis inicial H1 acerca de C.
    Procesar los restantes ejemplos: para cada Ei (i = 2, 3,...)
        Hacer
            Igualar la hipótesis actual Hi-1 con Ei. Sea el resultado una
                descripción D de las diferencias entre Hi-1 y Ei.
            Actuar sobre Hi-1 de acuerdo con D y de acuerdo a si Ei es
                un ejemplo positivo o negativo de C. El resultado es
                    la hipótesis refinada Hi sobre C.
        Fin
Fin
```

El resultado final de este procedimiento es H_n, que describe el concepto C aprendido a partir de los ejemplos dados. En esta tarea, *Arches* utiliza redes semánticas. Veamos cómo se representan los ejemplos y la secuencia de actualizaciones de las hipótesis con esta metodología.



El programa procede de la siguiente manera. El primer ejemplo E1 refleja la hipótesis usual de lo que es un 'arco'. El segundo ejemplo es un ejemplo negativo. Emparejemos E2 con H1: el resultado muestra la diferencia D, que no es otra que la relación extra, *toca*, en E2. Ya que ésta es la única diferencia, el sistema concluye que debería ser la razón de por qué E2 no es un

arco. El sistema actualiza ahora la hipótesis H1 aplicando el siguiente principio heurístico general de aprendizaje:

si
el ejemplo es negativo y
el ejemplo contiene una relación R que no está en la hipótesis actual H
entonces
prohibir R en H (añadir *no_tienen_que_R* en H)

Como se puede notar, el aprendizaje heurístico está definido como una regla condición-acción. El resultado de aplicar esta regla sobre H1 será una nueva hipótesis H2 que tiene como enlace extra *no_tienen_que_tocarse*, restricción que a partir de ahora debe tenerse en cuenta para diagnosticar correctamente qué es un arco. H2 resulta ser una hipótesis más específica que H1.

El siguiente ejemplo negativo está representado por la red semántica E3. El emparejamiento con H2, la hipótesis actual, revela dos diferencias: los dos enlaces *soporta*, presentes en H2, no están presentes en E3. Ahora el sistema aprendiz tiene que imaginar cuál de estas tres posibles disyuntivas se da:

(1) E3 no es un arco porque desapareció el soporte izquierdo
(2) E3 no es un arco porque desapareció el soporte derecho
(3) E3 no es un arco porque desaparecieron ambos soportes

Asumamos que la mentalidad del aprendiz es radical en cuanto a las transformaciones que provocan los cambios. En ese caso, requeriría la necesidad de ambos enlaces *soporta* y los convertiría en el requisito *tiene_que_soportar* en la hipótesis H3. La siguiente regla heurística de aprendizaje recoge esto.

si
el ejemplo es negativo y
el ejemplo no contiene una relación R que está presente en la hipótesis actual H
entonces
requiere R en la hipótesis nueva (añadir *tiene_que_R* en H)

De la aplicación de esta regla resulta una nueva hipótesis más específica que la anterior, al incluir dos enlaces *tiene_que_soportar*.

El último ejemplo, E4, es de nuevo un ejemplo positivo. El emparejamiento de la red de E4 a H3 muestra la diferencia: la parte superior es un triángulo en E4 y un rectángulo en H3. El aprendiz puede redireccionar el enlace *es_un* que hasta ahora era un rectángulo a una clase más general: *triángulo_o_rectángulo*. Supongamos que el aprendiz tiene como *background* de conocimiento del dominio a una taxonomía de conceptos en la que rectángulo y triángulo son clasificados como figuras del tipo *polígono_estable*. En este caso, el sistema aprendiz actualiza la hipótesis y obtiene H4.

Se ha podido observar que en el proceso de conformación de una hipótesis final, el sistema ha ido generalizando sucesivamente las hipótesis intermedias. La hipótesis final permite que la parte superior del 'arco' sea un trapecio, aunque ninguno de los ejemplos de arco suministrados mostrase esta característica al aprendiz.

El programa Prolog que implementa esta versión simplificada de *Arches* se puede consultar en Bratko (1990[2]). A ese programa podemos hacerle preguntas, que incluirán dos partes: la primera emplea el procedimiento predefinido *bagof*, que construye una lista con todas las posibles soluciones -incluídas las repetidas- a la pregunta `ejemplos(X)`. Sobre esa lista actúa la segunda parte de la interrogación, que incluye el procedimiento *aprender*, y cuya respuesta será el aprendizaje que el programa haga del concepto.

```
?- bagof(E, ejemplo(E), Ejemplos), aprender(Ejemplos, DescrArco).

E=_2,Ejemplos=[+objeto([parte1,parte2,parte3],[soporta(parte1,parte3
),soporta(parte2,parte3),es_un(parte1,rectangulo),es_un(parte2,recta
ngulo),es_un(parte3,rectangulo)]),objeto([parte1,parte2,parte3],[so
porta(parte1,parte3),soporta(parte2,parte3),toca(parte1,parte2),es_
un(parte1,rectangulo),es_un(parte2,rectangulo),es_un(parte3,rectangu
lo)]),objeto([parte1,parte2,parte3],[es_un(parte1,rectangulo),es_un
(parte2,rectangulo),es_un(parte3,rectangulo)]),+objeto([parte1,parte
2,parte3],[soporta(parte1,parte3),soporta(parte2,parte3),es_un(parte
1,rectangulo),es_un(parte2,rectangulo),es_un(parte3,triangulo)])],
DescrArco=concepto([parte1,parte2,parte3],[soporta(parte1,parte3),so
porta(parte2,parte3)],[es_un(parte3,poligono_estable)],[toca(parte1,
parte2)]) ;

No
```

Como se puede observar, la respuesta del compilador tiene dos partes: en la primera se listan los ejemplos de los que se ha servido para inducir el concepto; en la segunda, se describe el concepto. La descripción del concepto 'arco' se muestra en tres listas: la primera señala las partes constitutivas de un 'arco'-`([parte1,parte2,parte3])`; la segunda, las relaciones

admisibles entre esas partes y, si corresponde, su adscripción a una categoría -[soporta(parte1,parte3),soporta(parte2,parte3)],[es_un(parte3, poligono_estable)]-, y, la tercera, las relaciones prohibidas -([toca(parte1, parte2)])-. En consecuencia, *Arches* en un programa lógico inductivo que realiza un cierto aprendizaje del concepto de 'arco' a partir de los ejemplos suministrados.

6. Contribuciones de los programas al desarrollo de la actividad científica

Los ordenadores están dando los primeros pasos en el terreno del aprendizaje y, aunque se han obtenido algunos resultados interesantes, se circunscriben a áreas de conocimiento pequeñas. Hoy es impensable un sistema informático que, para cualquier dominio imaginable, aprenda a resolver los problemas que se le planteen en él -quizás esta cuestión tampoco tenga sentido para la Informática actual-. Pero los resultados que ya se han obtenido no son desdeñables. En el ámbito del aprendizaje automático, la programación lógica inductiva (PLI) se ha mostrado útil en la tarea de explicar y, por tanto, de conocer mejor lo que nos rodea.

La PLI tiene un papel en la conformación del conocimiento científico aportable por una máquina. Desde un paradigma inductivista y falsacionista de la ciencia, se puede establecer un parangón entre algunos aspectos del proceso de descubrimiento de las teorías científicas y los programas de aprendizaje. Por ejemplo, tanto un científico que intenta hacer un descubrimiento, como un programador que hace adquisición de conocimiento, usan observaciones o ejemplos del dominio de trabajo para proponer generalizaciones, aportando nuevo conocimiento. En ese proceso de generalización es útil actualizar experimentos que confirmen o falsen la teoría actual. Pues bien, la PLI favorece ambas tareas, ayudando al científico e ingeniero de conocimiento a corroborar o falsar sus conjeturas contrastando mecánicamente hipótesis y ejemplos relevantes que ayuden a tal propósito.

Un ejemplo de programa lógico inductivo que ya ha generado propuestas teóricas significativas para la comunidad científica de los bioquímicos es *Golem* (Muggleton y Feng 1990). *Golem* es una *shell* que detecta ciertas regularidades geométricas y que se aplicó, entre otros dominios, al campo de la estructura de las proteínas. Una proteína es una sucesión de aminoácidos unidos cada uno con el siguiente. Hay veinte aminoácidos distintos y según su disposición, conforman α-hélices y β-trenzas. Las α-hélices y β-trenzas permiten clasificar las proteínas en tres tipos: a) tipo -α; aquellas que sólo tienen α-hélices; b) tipo -β; las que tienen β-trenzas y c) tipo α/β,

aquellas que alternan α-hélices y β-trenzas. *Golem* se aplicó para predecir la estructura secundaria de las proteínas de tipo -α, una vez conocida su estructura primaria.

Para los bioquímicos no ofrece dificultad encontrar la estructura primaria de una proteína; esto es, la sucesión de aminoácidos que la componen. Más dificultad tiene conocer su estructura secundaria, que depende, esencialmente, de su forma tridimensional. Esta se puede determinar por medio de algunas técnicas caras y complejas (resonancia nuclear, rayos X, etc.) o utilizando *Golem*, que es capaz de predecir la estructura secundaria de las proteínas a partir del conocimiento que tengamos de su estructura primaria.

Golem usa la forma clausal del cálculo de primer orden, lógica implícita de Prolog, aunque por razones de eficiencia ha sido implementado en C, siguiendo la sugerencia metodológica, apuntada por Shapiro, de que Prolog es más un lenguaje para organizar conocimientos que una herramienta práctica de programación. Los antecedentes de *Golem* hay que buscarlos en el trabajo pionero de Plotkin, cuya aportación consistió en definir la generalización relativa mínimamente general, en acrónimo r.l.g.g. (*relative least general generalization*), inspirada en el principio de resolución y en el concepto de subsunción de J. Robinson, padre de la lógica clausal.

Golem se inicializa con un conjunto de ejemplos positivos y un conjunto de ejemplos negativos. Comienza tomando un ejemplo al azar de pares de ejemplos positivos y construye la r.l.g.g. de cada par. Escoge la r.l.g.g. que predice la mayoría de ejemplos verdaderos y menos ejemplos falsos respecto a un umbral predefinido. Una vez que se encuentra el par con la mejor r.l.g.g. (llamémosle M), *Golem* toma un ejemplo al azar de los ejemplos positivos todavía no predichos y forma la r.l.g.g. de M y de cada uno de los miembros de este nuevo ejemplo. Esas nuevas r.l.g.g. se evalúan como antes y el proceso continua hasta que no se produce ninguna mejora en la predicción.

Con la metodología descrita, *Golem* buscó iterativamente leyes o reglas con las que determinar el tipo secundario de un aminoácido particular por medio de sus propiedades. El programa Prolog de la *shell* de *Golem*, que descubre relaciones geométricas, atiende a las siguientes especificaciones:

- Datos usados en *Golem*:
  Necesita tres tipos de campos de entrada para construir las reglas:
  - Ejemplos positivos

- Ejemplos negativos
- Hechos básicos.

Con los ejemplos positivos se definen los márgenes de las mallas tipo EF (de elementos finitos). Cada margen se representa por el procedimiento:

```
margen(X, Y).
```

P. ej., *margen(b12, 1).* quiere decir que en el margen b12 hay 1 elemento-finito. Las cláusulas que describen los ejemplos negativos tienen la misma forma que la de los ejemplos positivos y la base de datos de ésta contendrá todas las combinaciones de nombres y números que no pertenezcan a los ejemplos positivos.

El campo que refiere a los hechos básicos contendrá todo el vocabulario que puede ser usado para describir hipótesis acerca de las mallas. Se estructura en cinco partes:

- Declaraciones
- Tipos de márgenes
- Condiciones de vecindad
- Cargas
- Representación geométrica.

La parte denominada 'declaraciones' define los modos y determinaciones de cada predicado; esto es, su formato (monádico, triádico...) y carácter (recursivo, no recursivo...). Así, la cláusula:

```
determ(malla(_,_), malla(_,_)).
```

indica que se permiten reglas recursivas para el predicado *malla.*

Los márgenes se clasifican en 12 tipos diferentes:

- Longitud grande
- Grande
- Menos grande
- No grande
- Circuito
- Medio circuito

...

- Agujero de medio circuito
- Agujero de un cuarto de circuito

Un ejemplo de hecho simple que define un tipo de margen es:

```
grande(a3).
```

Hay cuatro posibilidades para las condiciones limítrofes. Un margen puede ser:

- Libre
- Fijo por una cara
- Fijo por dos caras
- Rodeado

Hay, de forma similar, tres tipos para las cargas: Los márgenes pueden estar:

- No cargados
- Cargados por un lado
- Cargados en los dos lados
- Cargados por todas partes.

Hay la posibilidad de que un margen sea continuo y tenga al mismo tiempo una o dos caras cargadas, lo que quiere decir que algunos márgenes pueden ser descritos por uno o más predicados 'carga'.

La representación geométrica se necesita para establecer relaciones entre los márgenes. Las más importantes son las relaciones 'vecindad' y 'opuesto'. Esto es, se cree que los márgenes que son vecinos u opuestos se influencian mutuamente en la malla. La tercera relación importante es que algunas mallas tengan la 'misma longitud'. Los tres predicados tienen tres argumentos. Usándolos, se describieron mallas EF para tres diferentes estructuras de entrenamiento. El tamaño de los datos de entrada para *Golem* fueron los siguientes:

- 75 ejemplos positivos
- 618 ejemplos negativos
- 588 ejemplos del background

Esta concha o *shell* fue utilizada para descubrir regularidades geométricas en la estructura de las proteínas. Se seleccionaron como conjunto de entrenamiento doce de tipo α de estructura primaria conocida, a las que se le añadió el conocimiento básico de los expertos. *Golem* aprendió un pequeño número de reglas para predecir qué aminoácidos forman α-hélices.

Una de ellas, la número 12, dice que:

En la proteína A hay una α-hélice en la posición B,

si

El aminoácido en la posición B es grande, no es aromático y no es lisina.
El de B+1 es hidrofóbico y no es lisina
El de B+2 ni es aromático ni es prolina
El de B+3 no es aromático ni es prolina y es pequeño o polar
El B+4 es hidrofóbico y no es lisina
El B-2 no es prolina
El B-1 ni es prolina ni aromático

Esta 'ley de la naturaleza' no era conocida hasta entonces; *Golem* contribuyó, por tanto, a resolver un problema relevante en el ámbito de la bioquímica, al mostrar una relación causal entre la aparición de una sucesión determinada de aminoácidos y la estructura de la proteína.

No se puede decir que la ley descubierta por este programa tenga la misma generalidad y simplicidad que, por ejemplo, el enunciado de una ley de la física de Newton. *Golem* y otros sistemas de PLI son tramos de un largo camino por recorrer. Pero sería miope no reconocer que los ordenadores nos han ayudado ya, y nos ayudarán todavía más en un futuro, a establecer conjeturas interesantes sobre nuestro entorno; a tener más conocimiento de nosotros mismos y de lo que nos rodea.

7. Reflexiones filosóficas sobre el mecanicismo

Se han examinado ciertas repercusiones que han tenido algunos programas de inteligencia artificial para evaluar aspectos interesantes sobre la relación persona-computador en el ámbito de la Filosofía de la Ciencia. El objetivo ahora es examinar algunas tesis de naturaleza filosófica vinculadas al tema del mecanicismo y debatir brevemente las repercusiones que en esta discusión pueden tener los programas inductivos. La discusión sobre el mecanicismo nos retrotrae al tema de la indecidibilidad, discutido en este trabajo en el marco del problema de la correspondencia de Post. Para retomar el tema de la omnisciencia computacional nos referiremos, sólo informativamente, a los teoremas de incompletud de Gödel y al uso que de este resultado matemático han hecho algunos autores como Lucas o Penrose para sacar consecuencias acerca de la imposibilidad del mecanicismo; esto es, de explicar la mente humana en términos puramente mecánicos.

Para lo que nos ocupa, haremos referencia sólo al primer teorema de incompletud de Gödel, que se puede enunciar así: dado un sistema formal cualquiera S, tal que (1) S es consistente y (2) a partir de S se puede derivar una parcela suficientemente amplia de la aritmética; se puede encontrar una proposición indecidible p en S -esto es, una proposición tal que ni ella (p) ni su negación (no p) puedan ser probadas en S- y, sin embargo, se puede mostrar a p como un enunciado verdadero de la aritmética usando cualquier argumento informal externo a S.

Lucas usó el enunciado de este teorema para mostrar que el mecanicismo es falso, ya que de él dice seguirse que un observador humano puede mostrar que si S es consistente, p es verdadera, siendo p una proposición indecidible asociada a S -por tanto, una proposición acerca de la cual una máquina (una MT) no podría ofrecer diagnóstico alguno-. Gödel, en sus conversaciones con Wang, mostró un punto de vista diferente al de Lucas respecto a las consecuencias que se seguirían de su teorema. Se pueden resumir así: o bien las mentes humanas sobrepasan a todas las máquinas (para ser precisos, pueden decidir más cuestiones de teoría de números que cualquier máquina) o existen cuestiones de la teoría de números indecidibles para la mente humana. Esta tesis es, desde un punto de vista lógico, más rigurosa que la de Lucas y además, depende de una premisa mucho más fuerte: asumir que la mente humana puede decidir todas las cuestiones relativas a la teoría de números. Bien es cierto que Gödel requiere que sólo sean decidibles por la mente humana las cuestiones de la teoría de números, no todas las cuestiones de la matemática (p. ej., las preguntas de la teoría de conjuntos). Pero esto no exime de dificultades a su propuesta, porque los problemas de la matemática presentan cierta interdependencia. Como el propio Gödel mostró, algunos resultados en la teoría de conjuntos pueden tener influencia en la teoría de números. Si se consideran p. ej., ciertos axiomas de infinitud, se puede probar que tienen consecuencias externas al dominio de los números transfinitos, que es su ámbito de aplicación inmediato: bajo el supuesto de su consistencia, se puede mostrar que cada uno de los axiomas incrementa el número de proposiciones decidibles, como p. ej., en el campo de las ecuaciones diofánticas. En resumen, la tesis de Gödel sobre el mecanicismo es que la capacidad de la mente para tratar con el infinito -p. ej., en teoría de conjuntos- indica que no puede ser identificada con una máquina de estados finitos como lo es el cerebro.

La tesis de Lucas recibió objeciones. En este trabajo haremos referencia a las observaciones históricas de Putnam y Benacerraf y a otra más actual,

proveniente de los programas inductivos, que servirá de cierre a este artículo.

Putnam ofreció el siguiente contraargumento a la propuesta de Lucas. Supongamos que tenemos una máquina M y que *mi* cometido es hallar un enunciado p con el que pueda demostrar que si M es consistente, p es verdadero, aún siendo p indecidible a partir de M, por ser M consistente. Nada obsta para que M pueda probar que si M es consistente, p es verdadero. Pero la afirmación de la veracidad de p, que S no puede demostrar por ser consistente, tampoco *yo* puedo demostrarla, a no ser que pueda demostrar la incoherencia de M, asunto difícil si M tiene una cierta complejidad.

Otra objeción al argumento de Lucas proviene de Benacerraf. Para él, es plausible sostener que el cerebro es una máquina de Turing. Su argumentación se basa en que, ya que cada uno de nosotros no es capaz de saber su propio programa -en otras palabras, nadie puede conocer el sistema formal que le representa-, no se puede aplicar el método de Gödel para producir una proposición que es verdadera pero que no se puede probar en el sistema. Por tanto, el argumento de Lucas falla. Podría pensarse, como dice Grice, que uno sea incapaz de describir su propio programa, pero sepa describir el programa de otro, y, para él, mostrar una proposición indecidible. Bien, pero en ese caso -como argumenta Chihara-, no sería capaz de verbalizar su propio descubrimiento, lo que sería absurdo.

Penrose intentó salvar el argumento de Lucas refiriéndolo, no a un juicio individual, sino a una mente colectiva. Su argumentación discurre así: supongamos que en vez de referirnos a un algoritmo particular para *ver* como verdadera una proposición que una máquina no puede probar, aludimos a un algoritmo de algoritmos, a un algoritmo general, que resuma la competencia del matemático para construir pruebas. Al fin, todos los algoritmos particulares pueden ser comunicados y, después de una explicación, cualquier matemático debe poder manifestar su acuerdo o desacuerdo con cualquiera de ellos. Si dispusiéramos de tal algoritmo, basándonos en el teorema de Gödel, se extraería la consecuencia de que, aunque no podríamos conocer la verdad matemática, sí conseguiríamos *visualizarla.* Pero esto parece falso; por tanto, no puede haber tal algoritmo general. Esta conclusión le vale a Penrose para decir que hay algún tipo de conocimiento que no es algoritmizable y para el que, como se sabe, reclama una nueva física, que tienda un puente entre los dos paradigmas exitosos de la actualidad: la física mecánico-cuántica y el paradigma relativista.

Pero la tesis de Penrose parece también oportuna para sacar una conclusión distinta: ya que no puede haber un algoritmo general que resuelva cualquier problema en cualquier dominio -y, como se ha visto, no sólo por la razón por él apuntada-, lo oportuno no es plantear si el ser humano es o no superior a la máquina, sino reflexionar sobre si ambas entidades pueden cooperar en la resolución de problemas concretos. Se trata de considerar a la máquina no como un sistema estático, sino dinámico, y examinar qué consecuencias tiene ello para discutir el mecanicismo.

La tesis mecanicista se formula suponiendo que una mente particular, puede ser representada por un sistema formal S. A este supuesto se le aplica el primer teorema de incompletud de Gödel. Hay, sin embargo, en esta aplicación, un supuesto implícito: que S tiene que ser del mismo tipo que aquellos sistemas formales para los que se prueba ese teorema; esto es, S ha de constar de un conjunto de axiomas y los teoremas se deben derivar de los axiomas usando las reglas de la lógica clásica. Pero si quisiéramos representar la mente de un individuo por un sistema formal, este sistema seguramente tendría que ser de un tipo diferente de aquellos que aparecen en los textos matemáticos y para los cuales se prueban los teoremas de incompletud. Por una mente particular fluyen constantemente nuevas aportaciones que provienen del mundo externo. El sistema formal que la modelase no podría tener un conjunto fijo de axiomas, sino una colección dinámica, y se añadirían continuamente a medida que se ampliase el conocimiento. Además, S debería tener tanto reglas inductivas como deductivas. Las reglas inductivas permitirían establecer generalizaciones a partir de la nueva información adquirida. Además, algunas de las generalizaciones contravendrían a las antiguas, y S tendría que tener algunas reglas de inferencia para resolver tales conflictos y quizás restaurar la consistencia. ¿Se aplica todavía el argumento de Lucas a estos sistemas?

No está claro que el método de Gödel, desarrollado para sistemas formales estáticos donde tiene aplicación la lógica clásica se aplique también a sistemas formales dinámicos con reglas de inferencia inductivas o, en general, no clásicas. Se podría salvar esta dificultad relativizándola a sistemas dinámicos tomados en un momento particular t. Supongamos que S es una representación exacta de mi mente en el instante t. Para un tiempo u, u>t, puedo decir que una sentencia indecidible p en t es verdadera, con lo que el individuo en cuestión mostraría su *superioridad* sobre S. Pero si se prescindiese del tiempo, el momento relevante para ese individuo sería ahora u y para ese instante, valdrían las consideraciones anteriores.

Si en vez de referirnos a un sistema formal lo hacemos a una MT, las dificultades serían las mismas. Sabemos que, aún cuando las reglas inductivas o no clásicas sean complejas, éstas pueden ser simuladas por una máquina de Turing y los axiomas resultantes de las entradas sensoriales pueden ser escritos como datos de la cinta de entrada. *Golem* es un ejemplo de esto. Supongamos que se diseña una MT particular para simular el comportamiento de un cerebro A. Las aferencias sensoriales de A se corresponderán con lo que se escriba en la cinta de entrada de la MT. El procesamiento de esta información corresponde al cambio de estado de la máquina. Cada estado global de A corresponde a un estado de la máquina y el programa de MT corresponde al proceso neurológico que lleva de una configuración cerebral a otra. ¿Podría ser representado todo esto por un sistema formal?. Parece que si, porque es conocido que las máquinas de Turing pueden ser especificadas de términos de números y de operaciones entre números. Supongamos entonces que podemos aplicarle el método de Gödel al sistema resultante y obtener una proposición p indecidible. ¿Qué quiere decir esto?. Que la habríamos obtenido no con la representación original -en términos de MT-, sino con el sistema numérico en el que se codificase ésta. Por tanto, no se podría aplicar el argumento de Lucas.

Se podría argüir que la consideración de sistemas inductivos de aprendizaje es irrelevante para el tema que nos ocupa en este apartado. Pero no es así, porque la cuestión crucial no es la de si las personas son superiores a las máquinas o si éstas son más capaces que aquellas. En términos pragmáticos, esta claro que para algunos propósitos la mente humana es superior a la máquina -p. ej., tareas que conllevan razonamiento ordinario- y para otros, es la máquina la que supera a los humanos -p. ej. en la empresa de realizar operaciones aritméticas-. La cuestión importante es decidir si son iguales o no y, caso de que la respuesta fuese negativa, indicar cuál es la posición y el papel de cada una de estos agentes en el juego cooperativo de hacer avanzar el conocimiento. Parece claro que máquinas y mente humanas no son iguales: la mente es fruto de un proceso histórico-evolutivo del que carece la máquina y su sustrato físico es biológico, no de silicio. Hay otra diferencia que aún siendo de perogrullo es importante: es el humano el que creó la máquina y no al revés. El humano causó la computadora, no a la inversa.

¿Qué aportan, entonces, los programas inductivos como *Golem* a esta discusión? Más que arrojar luz sobre si las mentes superan o no a las máquinas, los programas de aprendizaje automático enseñan como ambas entidades pueden cooperar en la generación de conocimiento científico. Re-

cordemos que para que *Golem* funcionase se necesitaba algún fondo o *background* de conocimiento. Si el programa descubre alguna ley hasta ahora no conocida, la ofrece al experto humano, que debe integrarla en el conocimiento existente. Al hacerlo surge otro fondo diferente al anterior, que quizás conduzca a una modificación del programa. De esta manera, el conocimiento avanza como un proceso de interacción persona-máquina, en el que la contribución humana resulta vital, pero también la máquina presta ayuda, como instrumento que permite desarrollar e incluso proponer conocimiento inédito. Con ello, el humano logra incrementar la sabiduría que tiene sobre sí mismo y sobre lo que le rodea, usando a la máquina como un instrumento extra de aporte de conocimiento.

Un ámbito interesante para investigar la relación cooperativa persona-máquina es la teoría de juegos. Un jugador mecánico tendrá un conocimiento exhaustivo de aquellas características del juego que sean recursivas; entre ellas, de la función de pago asociada a las reglas de juego. Esto quiere decir que los jugadores algorítmicos son capaces de justificar razonadamente su comportamiento, ofreciendo buenas razones. Y no parece que en este contexto la tesis de Lucas sea relevante, porque lo sería sólo si quisiéramos que los jugadores fuesen capaces de justificar no sólo lo que hacen, sino también lo que conocen. Pero puede que, en este contexto, no sea razonable exigir esto ni a las máquinas, ni a las personas.

Agradecimientos. Gracias a los evaluadores anónimos por sus interesantes sugerencias. Ayuda económica a cargo del proyecto PGIDT99PXI10502B de la Xunta de Galicia.

BIBLIOGRAFIA

Benacerraf, P.: 1967, 'God, the Devil and Gödel', *Monist* 51, 9-32.

Binmore, K. & Shin, H.S.: 1992, 'Algorithmic knowledge and game theory', in C. Bicchieri & M.L. Dalla Chiara (eds.): *Knowledge, Belief and Strategic Interaction*, Cambridge Univ. Press, pp. 141-154.

Bratko, I.: 1990[2], *Prolog Programming for Artificial Intelligence*, Addison-Wesley.

Chihara, Ch.S.: 1972, 'On Alleged Refutations on Mechanism using Gödel's Incompleteness Results', *Journal of Philosophy* LXIX, Nº 17, 507-26.

Driessen, A. & Suárez, A. (eds.): 1997, *Mathematical Undecidability, Quantum Nonlocality and the Question of the Existence of God*, Kluwer Academic Publishers.

Gillies, D.: 1999, *Artificial Intelligence and Scientific Method*, Oxford University Press.

Hopcroft, J.E. & Ullman, J.D.: 1979, *Introduction to Automata Theory, Languages and Computation*, Reading, MA, Addison-Wesley.

Lucas, J.R.: 1961, 'Mentes, máquinas y Gödel', in A.R. Anderson (ed.): 1984, *Controversia sobre mentes y máquinas,* Barcelona, Tusquets, pp. 69-94.
McCall, S.: 1999, 'Can a Turing Machine know that the Gödel Sentence is true?', *Journal of Philosophy* 99/9610, 525-32.
Muggleton, S. (ed.): 1992, *Inductive Logic Programming,* Academic Press.
Penrose, R.: 1991, *La nueva mente del emperador,* Madrid, Mondadori.
Putnam, H.: 1961, 'Mentes y máquinas', in A.R. Anderson (ed.): 1984, *Controversia sobre mentes y máquinas,* Barcelona, Tusquets, pp. 113-150.
Robinson, J. A.: 1965, 'A Machine-oriented Logic Based on the Resolution Principle', *J. of the Computing Machinery* 12, 23-41.
Shönning, U.: 1989, *Logic for Computer Scientist,* Birkhäuser.
Turing, A.M.: 1950, 'Maquinaria computadora e inteligencia', in A.R. Anderson (ed.): 1984, *Controversia sobre mentes y máquinas,* Barcelona, Tusquets, pp. 11-50.
Trillas, E.: 1998, *La Inteligencia Artificial. Máquinas y personas,* Madrid, Ed. Debate.
Wang, H.: 1996, *A Logical Journey, From Gödel to Philosophy,* The MIT Press, Ch. 6.

Alejandro Sobrino es profesor titular del Departamento de Lógica y Filosofía Moral de la Universidad de Santiago de Compostela (USC), en la que enseña *Teoría formal de lenguajes* y *Programación lógica.* Ha sido vocal de la Sociedad Española de Tecnologías y Lógica Fuzzy y es Secretario de la Facultad de Filosofía de la USC. Sus líneas de trabajo actuales son: lenguajes fuzzy, inducción y aprendizaje automático y aspectos filosóficos de la I.A.