JUEGOS CON DADOS, CÓMPUTOS NUMÉRICOS.
Cómo puede ser calculado un conjunto numérico aleatorio

Enrique ALONSO
Universidad Autónoma. Madrid

1. Presentación del problema

Al amparo de un impresionante éxito material, la Teoría de la Computación ha conseguido asentarse en el plazo de tan sólo 60 años como una de las disciplinas científicas más sólidas y activas. La pronta resolución de sus conflictos fundacionales ha permitido una rápida evolución cuyo resultado es una disciplina muy alejada en la actualidad de las preguntas y objetivos que dieron lugar a su nacimiento en la década de 1930. La tarea de revisar sus fundamentos parece quedar, por tanto, para un reducido grupo de investigadores aparentemente inmunes al impacto negativo que por lo común producen las ciencias formales entre sus pares.

El esfuerzo de explicarse a uno mismo, y hacer entender a otros, la estructura interna de esta disciplina, en lo que con propiedad constituye una especie de exposición *abstracta* o *general* de la Teoría de la Computación, impone, sin embargo, una tarea de revisión cuyo resultado es a menudo la localización de fisuras en una estructura uniforme de otro modo. Este ensayo está dedicado a describir y analizar uno de esos conflictos cuyo origen se encuentra en nuestro afán por atribuir al sano sentido común consecuencias que en realidad, sólo son aceptables violentando nuestras intuiciones.

Sin más dilación, ¿qué queremos decir cuando calificamos un conjunto numérico finito obtenido, esto es lo importante, por un procedimiento aleatorio, el calificativo de *computable* —equivalentemente, recursivo—? En otras palabras, ¿cómo es posible que una entidad obtenida por un método tan arbitrario como el descrito pueda, no obstante, recibir un calificativo —computable— re-

ÉNDOXA: Series Filosóficas, n.° 10, 1998, pp. 21-47. UNED, Madrid

servado en apariencia a objetos ligados a un procedimiento efectivo de cálculo? No reconocer en esto una cato de violencia sobre nuestras intuiciones informales sólo puede responder a una fe digna, tal vez, de mejor causa.

Haciendo gala de una notable suficiencia, la Teoría de la Computación se limita en este caso a guiarnos hasta la definición de *conjunto recursivo* y de ahí a una respuesta definitiva: dado que todo conjunto *finito* es recursivo, la computabilidad de cualquier conjunto finito *arbitrario* se sigue como corolario. Aprovecho la ocasión para recordar al lector que un conjunto de enteros positivos es *recursivo* si y sólo si:

[1] i. su función característica χ es recursiva[1], o bien,

ii. tanto Γ como su complementario constituyen el dominio de sendas funciones recursivas φ y ψ.

Esta respuesta no dice, sin embargo, de qué forma podemos integrar coherentemente dos calificativos tan opuestos como los de *computable* y *arbitrario* reduciendo así la tensión impuesta sobre nuestras intuiciones preformales.

Aunque en una vista preliminar la Teoría de la Computación puede hacer gala de una relación especialmente fructífera con la noción de conjunto[2], un simple uso no establecido de los términos puede ponernos ante casos que, como el que nos ocupa, son difíciles de interpretar. Lo que parece provocar tensión al hablar de conjuntos arbitrarios y computables es la implícita multiplicidad de significados que desde un punto de vista informal atribuimos a la computabilidad de un conjunto. Los dos siguientes representan, posiblemente, extremos significativos:

[2] i. Un conjunto es computable si existe un procedimiento efectivo para su construcción.

---

[1] Está en el ánimo de este ensayo no presuponer en el lector un conocimiento detallado de los contenidos de la Teoría de la Computación. No obstante, sí parece conveniente disponer de unos mínimos rudimentos. Sea como fuere, me apresuro a aclarar que en este escrito una *función recursiva* representa el mismo concepto formal que el de *función recursiva parcial* y que el de *función Turing-computable*, o simplemente computable.

[2] Por ejemplo, a través de clasificaciones como las que dan lugar a la noción de *grado de insolubilidad*.

ii. Un conjunto es computable si existe un procedimiento efectivo para establecer, dado un entero positivo, si éste pertenece o no al conjunto en cuestión.

Aunque hemos aprendido a identificar estas interpretaciones informales de la computabilidad —una vez conocida su equivalencia formal— es evidente que no apuntan a lo mismo. En [2.i] se apela a la existencia de un procedimiento capaz de *generar* o *definir* un conjunto con independencia de cualquier dato que pudiera obtenerse del conocimiento previo de su extensión o de cualquier otra condición explícita previa acerca de aquella. En [2.ii], por el contrario, se interpreta la computabilidad de un conjunto como la capacidad de reconocer de manera efectiva su extensión no importa a través de qué medios. Esta última interpretación obliga a reconocer como un modo legítimo de establecer la computabilidad de un conjunto contar con un listado de sus elementos a partir del cual juzgar la pertenencia de aquellos que en cada caso se analizan. Es evidente que este procedimiento sólo resulta efectivo si el cardinal del conjunto en cuestión es finito.

Aunque, como ya he dicho, la Teoría clásica de la Computación pretende dar cuenta de ambos planteamientos, es bastante obvio que su posición resulta mucho más coherente con la versión de la computabilidad que se ofrece en [2.ii] que con la expuesta en [2.i]. Calificar un conjunto de *arbitrario* parece indicar la inexistencia de un procedimiento efectivo asociado a su construcción y, por tanto, llevaría en buena ley a rechazar su recursividad entendida en el sentido de [2.i]. Constatar la finitud del cardinal de un conjunto sugiere por sí misma la existencia de un método para determinar si un entero positivo pertenece a dicho conjunto, lo cual basta para garantizar su recursividad en el sentido de [2.ii]. Al sostener que todo conjunto arbitrario finito es recursivo se hace evidente que la interpretación favorita de la computabilidad es aquella que se desprende de [2.ii].

Mi objetivo aquí no es evaluar cuál de los dos sentidos es el *correcto*. El problema que se afronta no se dirige a justificar una interpretación más estricta de la computabilidad capaz de corregir los pretendidos desmanes de la versión hasta ahora oficial. Mi sospecha no apunta a la revisión de la interpretación matemática de la noción intuitiva de *tarea efectiva*, lo que equivaldría a una nueva disputa sobre la Tesis de Church, sino a la pretendida *univocidad* de la noción intuitiva de tarea efectiva. Aplicamos procedimientos formales de análisis que

ignoran, o no hacen explícitos, aspectos específicos de las tareas que en cada caso se proponen. Esto es lo que parece suceder en ocasiones al estudiar la computabilidad de un conjunto desde el punto de vista de la Teoría de la Recursión: no parece haber un único modo de orientar la cuestión y por tanto de resolver el problema de la computabilidad de conjuntos numéricos.

Un conjunto arbitrario finito es computable siempre que la tarea asociada a esa declaración sea la de establecer la pertenencia de un entero positivo a su extensión, sin embargo, si lo que se considera en ese caso es la tarea de construcción del conjunto, entonces y sólo entonces, es cuando el análisis estándar se vuelve inadecuado. Es ahí dónde debe situarse mi reconocimiento de la adecuación general de la Teoría clásica de la Computación por lo que se refiere a la interpretación de la efectividad, al tiempo que mi rechazo ante la pretendida univocidad de la noción intuitiva de tarea efectiva. El éxito abrumador de la Teoría de la Computación parece depender a veces de la posibilidad de una interpretación ambigua de las tareas asociadas a aquellos problemas que le competen.

Mi propósito en este ensayo es intentar identificar una clase de funciones computables que puedan emplearse en contextos asociados a la construcción de conjuntos sin especial temor a la aparición de consecuencias extrañas. Aunque el planteamiento es completamente afín a los objetivos y estrategias de la llamada Matemática constructiva, las técnicas que habré de emplear no lo son en igual medida. Debo reconocer en este punto mi deuda con buena parte del espíritu que anima la denominada *Random Information Theory* atribuida al ingenio Kolmogoroff, Chaitin, y Solomonoff entre otros. Una exposición más pormenorizada de mis posiciones requeriría, a buen seguro, una lectura de ciertas contribuciones y tesis fundamentales de estos y otros autores. Es obvio, no obstante, que el detenimiento y detalle necesarios para ello queda fuera de toda prudencia.

2. Construcciones y conjuntos arbitrarios

Con frecuencia vemos cómo un conjunto se define a partir de una función numérica supuestamente conocida. De hecho, este es el modo más directo de establecer una conexión entre la noción de conjunto y sus propiedades y atributos *qua* conjunto y la Teoría de la computación. Pero el uso de funciones en la definición de conjuntos no pone en pié de igualdad ambos objetos matemá-

ticos: lo que realmente pretendemos al definir un conjunto a través de una función es ofrecer un procedimiento capaz de establecer la extensión del conjunto así analizado. Negar este punto supondría, así me lo parece, atentar contra buena parte de la práctica matemática pasada y contemporánea. Este tipo de definiciones, afines a lo que propiamente es la construcción de una cierta entidad, un conjunto en este caso, imponen un contexto en el que se hace preciso respetar ciertas reglas: las relativas a qué es una buena definición. Una de ellas dice, por ejemplo, que el *definiens* no debe figurar como parte propia del *definiendum.* Aunque es realmente difícil establecer un criterio satisfactorio capaz de determinar en qué casos una función computable sirve a los efectos de construir realmente un conjunto, sí es posible determinar con cierta inmediatez cuándo no lo hace. Considérense un conjunto Γ y una función φ computable y totalmente definida. La siguiente identidad constituye, al menos a partir de los datos disponibles, una aparente construcción de Γ:

[3] $\Gamma = \{x / \varphi(x) = 0\}$.

Ahora bien, si a continuación se procede a definir $\varphi(x)$ de modo que

[4] $\varphi(x) =$ 0 syss $x \in \{a_0,...a_i,...a_n\}$, donde $\{a_0,...a_i,...a_n\}$ es precisamente Γ
1 en otro caso,

la plausibilidad de [3] como una definición, digamos constructiva, de Γ desaparece de inmediato. La combinación de [3] y [4] sólo se puede considerar, y eso en el mejor de los casos, como un innecesario rodeo en la presentación explícita de Γ.

Esta situación, sin embargo, es la única que cabe esperar en conjuntos finitos arbitrarios. En otras palabras, los conjuntos arbitrarios finitos parecen ser aquellos que no pueden ser definidos de manera constructiva mediante el uso legítimo de alguna función computable.

Es posible que el lector menos avezado en las sutilezas de la Teoría de la Computación piense que la solución a nuestro problema consiste simplemente en impedir la presencia de una función claramente improcedente en [4]. Para que el mecanismo de construcción de conjuntos que se desprende de la com-

binación de [3] y [4] resulte adecuado bastará entonces con excluir en [4] funciones que requieran una mención explícita a los elementos de un cierto conjunto. Por desgracia, nos movemos en un terreno en el que una declaración de principios como la que se acaba de hacer difícilmente puede contar como una solución del problema. La identificación de información numérica en la definición de una función no es algo que quepa discutir como un asunto meramente nominal. Por el contrario, es imprescindible situar el problema bien dentro del cuerpo del cuerpo de dogmas que forma la Teoría de la Computación. No creo posible progreso alguno en la situación sin un estudio de los mecanismos explícitos —funciones recursivas, y preferentemente, máquinas de Turing— que determinan la forma en que cierta información numérica puede formar parte de la definición de una función. Sólo así podría establecerse algún criterio razonable que permitiera distinguir los usos legítimos de los no legítimos en aquellas definiciones basadas en la combinación de [3] y [4]. Cabe suponer que una función computable definible sin el concurso esencial de cualquier colección de enteros positivos nunca daría lugar a conflictos definicionales como el se viene comentando. Esto me lleva a proponer como primera hipótesis la siguiente definición:

[5] Un conjunto numérico es arbitrario si y sólo si toda máquina de Turing capaz de computar su función característica codifica información explícita acerca de su extensión en su tabla o gráfico[3].

Como es evidente [5], no va a sobrevivir a las más elementales objeciones, no obstante, me atrevo a pedir del lector que asuma con ingenuidad lo que de razonable hay en él mientras la discusión avanza.

3. Limitaciones generales y máquinas de Turing

Cualquier intento de analizar una clase de funciones computables propiamente incluida en la clase de las funciones Turing-computables cae de inmediato bajo el alcance del conocido Teorema de Rice, cuyo enunciado establece la imposibilidad de definir una clase *recursiva* de funciones menor que la pro-

[3] Cfr. *infra*, def. 1 y 2.

pia clase de todas las funciones computables. Este teorema parece haber ejercido una influencia considerable a la hora de proseguir investigaciones que de algún modo se ven afectadas por sus términos, y la nuestra, ciertamente, parece una de ellas.

Aunque [5] tiene como referente inmediato la noción de Máquina de Turing y no las funciones que éstas computan, no por ello se consigue evitar el impacto del teorema mencionado. Como es sabido, existen clases de máquinas que sí pueden ser definidas recursivamente, siempre y cuando, claro está, no definan ellas mismas una clase de funciones computables significativa[4]. [5] no figura, sin embargo, entre ellas. La mención a la clase de máquinas que computan una misma función, la función característica de un conjunto numérico Γ, sitúa el problema más allá de los límites de la decidibilidad: la clase de máquinas que es preciso analizar comprobando si codifican o no información numérica explícita no es recursivamente definible[5].

Estos comentarios aconsejan reformular [5] eliminando cualquier mención a una clase de máquinas de la cual tenemos tan poco control. Así pues,

[6] Diremos que un conjunto numérico (tiene una *construcción aceptable*[6] syss su función característica es actualmente computada por una máquina que no codifica información numérica arbitraria en su definición.

Es evidente que todo conjunto numérico no arbitrario —en el sentido que este término adquiere en [5]— tiene al menos construcción aceptable, aunque

---

4 En realidad basta con que esa clase de funciones no sea cerrada bajo identidad.

5 Esto es así en la medida en que dicha clase estaría formada por todas las máquinas que computan la función característica de un cierto conjunto Γ, clase que, obviamente, es cerrada bajo identidad y por tanto cae bajo el alcance del Teorema de Rice. Supuesto además que se hubiera identificado en la máquina que computa un cierto conjunto numérico la existencia de información numérica arbitraria, el efecto del Teorema de Rice obligaría a reconocer la inexistencia de un procedimiento mecánico que juzgue si tal información es *eliminable* dando lugar a otra máquina equivalente a la anterior.

6 Tal vez hubiera sido más adecuado hablar de definiciones de tipo *constructivo* de un conjunto, sin embargo, esto me hubiera situado mucho más cerca de los móviles de la Matemática constructiva de lo que realmente deseo estar. Aunque las motivaciones puedan ser en muchos casos afines, cualquier uso de términos como *construcción* o *constructivo* tienen lugar en un contexto por lo demás clásico.

como es obvio, la no disponibilidad actual de una no permite ya calificar al conjunto definido de arbitrario. En lo sucesivo me centraré en la identificación de este tipo de definiciones cuya estructura es, en última instancia, la que más importa conocer.

La necesidad de operar con aspectos concretos de la arquitectura interna de las máquinas de Turing hace conveniente fijar alguna terminología, por lo demás sobradamente conocida.

Definición 1: Una *instrucción* es cualquier tupla del tipo $q_i x a q_j$ donde,

i. q_i es un entero positivo y q_j es cualquier número natural incluido el *0*.

ii. $x \in \{0,1\}$

iii. $a \in \{0,1,I,D\}$.

Definición 2: Una *máquina de Turing* es cualquier colección finita de instrucciones que satisface además el siguiente requisito:

— Nunca es posible encontrar dos instrucciones i_j, i_k distintas cuyos dos primeros componentes $q_j x$, $q_k x$ coincidan.

Suponemos, además, que una máquina actúa siempre sobre una cinta ilimitada en ambos sentidos y dividida en unidades discretas que denominamos *celdas*. La cabeza lectora de la máquina se desplaza de celda en celda —una por vez— escrutando el contenido de cada una de ellas. De la definición 1 se desprende que los únicos símbolos que pueden figurar en una celda son *1* y *0*. A esto se añade la estipulación de que cada celda debe contener uno y sólo uno de ellos. Esto permite interpretar el *0* como indicativo o sinónimo de que la celda se halla vacía y el *1* como indicativo de celda ocupada. Una vez hechas estas aclaraciones, la interpretación de nuestras instrucciones se hace evidente. Así, por ejemplo, la pareja de instrucciones i_1=1012, i_2=11R1 provocan que la máquina M que escruta una cierta celda hallándose activo el estado *1* proceda a imprimir el símbolo *1* sobre la celda si el contenido de ésta es *0* activando a

continuación el estado *2*. Si, por el contrario, su contenido es *1* entonces se desplazará una celda a la derecha activando de nuevo el estado *1*. Para referirnos al listado de instrucciones que definen una máquina utilizaremos frecuentemente el término *tabla*. El requisito que la def. 2 añade a un listado de instrucciones para hacer de éste una auténtica tabla de máquina se refiere simplemente a la univocidad con que deben determinarse las acciones que siguen al acto de escrutar el contenido de una celda. Así, nunca admitiremos que una vez conocido el contenido de una celda la acción a ejecutar deba sea elegida de entre varias posibles —situación sólo atribuible a la existencia de instrucciones que comparten sus dos primeros componentes difiriendo en el resto. Finalmente, sólo resta aclarar que el numeral *0* ha sido excluido como primer elemento de una instrucción al actuar como indicativo de una orden de parada. Cualesquiera otras convenciones que sea oportuno adoptar se explicarán cuando resulte preciso.

4. Información numérica explícita y máquinas de Turing

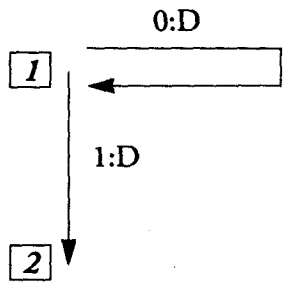
La forma más indicada de analizar la presencia de información numérica en una máquina de Turing es mediante sucesivas aproximaciones que aquí denominaré *modelos*. Cada uno de ellos investiga una hipótesis al respecto y pone a prueba su adecuación al caso. Presentaremos un total de cuatro, de los cuales, sólo el último parece soportar bien todas las objeciones razonables.

Primer modelo

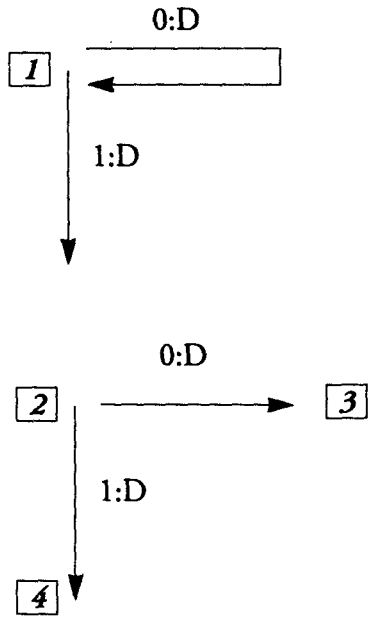
Cualquiera que haya experimentado con el diseño de máquinas de Turing sabe que el medio más directo de codificar información numérica explícita está de algún modo asociado a la repetición de ciertas instrucciones en la tabla de una máquina. Por ejemplo,

[7]

i. 10D1
11D2



ii. 10D1
11D2
20D3
21D4



Para mayor claridad, convendrá considerar que la celda sobre la que se halla inicialmente emplazada la cabeza lectora se encuentra ocupada por el símbolo *0* —o como diremos con frecuencia, se encuentra vacía. Según eso, la máquina [7.i], hasta donde muestra su tabla, recorre la cinta hacia la derecha hasta encontrar la primera celda ocupada —ocupada por el símbolo *1*. Una vez hallada, se activa la subrutina que se inicia en el estado *2*. La máquina [7.ii] actúa del mismo modo —de hecho comparten un fragmento de sus tablas— para exigir a continuación la presencia inmediata de otra celda más también ocupada. En caso de no darse esta combinación, la máquina [7.ii] no activará, hasta don-

de sabemos, la subrutina que se inicia en *4*. Es fácil imaginar, en definitiva, cómo extender la tabla de [7.ii] dando lugar a máquinas que exijan secuencias ocupadas de longitud creciente repitiendo simplemente el formato básico del estado *2*.

Mientras que [7.i] identifica simplemente la presencia de alguna celda ocupada en un recorrido ilimitado hacia la derecha de la cabeza lectora [7.ii], y sus posibles extensiones exigen la presencia de secuencias de longitud mayor o igual que *2* y mayor o igual que *n* para activar una cierta subrutina. Cualquier evaluación razonable de esta situación reconocerá que [7.ii] muestra una conducta determinada por información numérica explícita que no se observa, sin embargo, en [7.i]. La claridad de este ejemplo permite asociar esa conducta a la presencia de ciertas instrucciones en las tablas respectivas. Como ya se ha indicado, la repetición de un patrón que podemos resumir como:

[8] i0Dj
i1Dk

donde $i \neq j$, $i \neq k$ y $j \neq k$, siendo a su vez la variable *k* la que encabezaría el par de instrucciones de la siguiente instancia de [8].

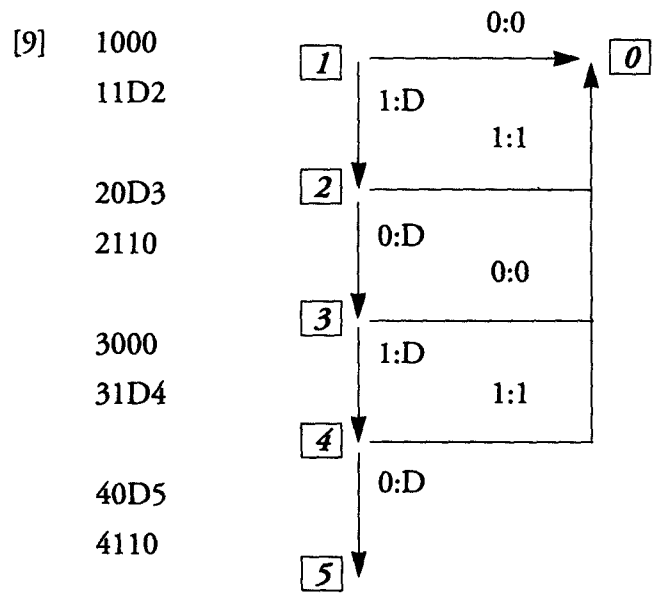
Si realmente se pudiera conectar la codificación de información numérica en una máquina a la presencia de un patrón del tipo representado en [8], la respuesta a la pregunta que indirectamente se propone en [6] no podría presentar mejor cariz. Puesto que toda máquina viene definida por una tabla finita, y [8] es un patrón identificable en cualquier tabla, las máquinas que codifican información numérica explícita en sus tablas podrían ser definidas recursivamente. Sin embargo, y por desgracia, la cuestión que se discute presenta complicaciones que van mucho más allá de esta esperanzadora conjetura.

De un lado, resulta evidente que la identificación de una secuencia de n-celdas ocupadas, pese a estar asociada al patrón presentado en [8], admite complicaciones que resulta imposible sistematizar. Basta considerar cómo cualquier máquina que intercale entre las dos instrucciones que componen [7.ii] una subrutina cuyo efecto sea el de dejar inalterado el resultado final del cómputo que sucede entre ellas debería ser admitida entre aquellas que codifican información numérica explícita. La identificación de una de estas máquinas seguiría ahora

una estrategia sustancialmente distinta a la sugerida líneas atrás. De hecho, se haría preciso contar con un procedimiento efectivo capaz de determinar si una subrutina intercalada entre dos instrucciones repetidas se comporta como una subrutina eliminable para todo posible contenido de la cinta de cálculo. Como es obvio, esta clase de subrutinas no es recursivamete definible, lo cual resta interés a la hipótesis que liga la codificación de información numérica a la simple repetición de instrucciones.

Segundo modelo

Más importante que esta debilidad, posiblemente apreciable en cualquier criterio que merezca la pena considerar, lo es la propia limitación del planteamiento que se discute. La identificación de una secuencia de n-celdas ocupadas en una cinta no deja de ser, por muchas convenciones adicionales que se puedan adoptar, una instancia de un fenómeno más general: la conexión entre la conducta de una máquina y la presencia de una subsecuencia finita particular de celdas. Cualquier máquina en la que se asocie una conducta específica a la identificación de una cierta secuencia binaria en particular sobre la cinta de cálculo estará codificando, se viene a afirmar, información numérica explícita. Así, la máquina



activa el estado *5* sólo si la cinta de cálculo sobre la cual se encuentra emplazada la cabeza lectora presenta la subsecuencia binaria «1010» hacia la derecha de su posición inicial, deteniéndose de otro modo. En este caso, el patrón a que responde [9] presenta una estructura considerablemente más difícil de evaluar. Si hubiera que señalar uno tal vez lo más adecuado sería identificar la instrucción típica

[10] ixDj, donde $x \in \{0,1\}$ y $i \neq j$.

Sin embargo, la localización de repeticiones del modelo presentado en [10] no parece penetrar todo lo preciso en el modelo de codificación de información explícita que ahora se estudia. En definitiva, estamos reinterpretando la presencia de información explícita en la tabla de una máquina como un fenómeno directamente asociado al tipo de condiciones que deben darse para que la rutina que ejecuta dicha máquina logre activar un estado *m* cuando se halla en otro estado *n*. En concreto,

[11] una máquina codifica información explícita en su tabla si la única forma de activar un estado *m* hallándose activo otro estado *n* de esa misma máquina tiene lugar cuando la cinta de cálculo presenta alguna combinación finita de celdas de entre una colección finita de opciones.

Aunque [11] parece acercase a lo que podría ser una respuesta satisfactoria, basta una observación elemental para desbaratar esta impresión. Secuencias del tipo «10», o «01» constituyen instancias indiscutibles de subsecuencias finitas como las que se mencionan en [11]. La primera de ellas ocurre, de hecho, como subsecuencia inicial de cualquier secuencia acotada por la izquierda de celdas ocupadas en un hipotética cinta de cálculo. La segunda lo hace, a su vez, como subsecuencia final de toda secuencia acotada por la derecha. Según eso, las subrutinas que identifican el comienzo o final de secuencias acotadas de celdas ocupadas en una cinta de cálculo instancias de subrutinas asociadas a procesos de codificación de información explícita, lo que es dudosamente aceptable.

Considerar excepciones basadas en la longitud de las subsecuencias evaluadas, o restricciones ligadas al contexto en que una subsecuencia ocurre dentro de una cinta de cálculo podría mejorar sustancialmente la aplicabilidad del criterio descrito en [11]. No obstante, una solución basada directamente en [11] capaz de eliminar toda casuística parece bastante difícil de imaginar.

La relación propuesta entre codificación de información numérica explícita y las condiciones para que una rutina active un estado *n* desde otro estado *m* es, no obstante, lo mejor que se ha obtenido en el curso de esta discusión. Sucede, simplemente, que el modo en que he interpretado esta presunta conexión es aún excesivamente impreciso y algo restrictivo a la vez.

Tercer modelo

Si se toma como unidad de análisis la noción de *estado*, se ve que entre las muchas formas existentes de clasificar y analizar aquellos que se obtienen a partir de instrucciones como las presentadas en la definición 1, existe una especialmente relevante para lo que aquí se discute. Se puede decir, por ejemplo, que un estado *n* es de tipo *variable* si es capaz de permanecer activo mientras la cabeza lectora procede a escrutar un número indefinido aunque finito de celdas sobre la cinta de cálculo —esta definición requiere que se asuma que toda cinta contiene un número finito distinto de *0* de celdas ocupadas. Si sucede, por el contrario, que un estado *n* sólo es capaz de permanecer activo un número finito[7] de celdas antes de dar paso a la activación de otro estado n' diremos que se trata de un estado *discreto*. Para que la lista sea exhaustiva, hay que incluir aquellos estados que obligan a la cabeza lectora a escrutar una cantidad infinita de celdas, y a los cuales denominaremos *degenerados*. Parece evidente, además, que los estados de tipo variable deben constituir instancias del siguiente modelo o patrón característico:

[12] nx (I/D)n

ny (I/D/0/1)m, donde $x \neq y$ $n \neq m$.

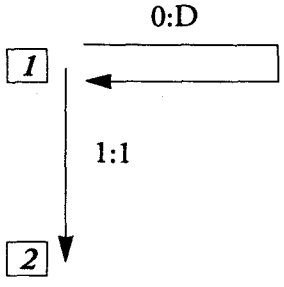
---

7 En nuestro caso, y bajo las convenciones y vocabulario adoptado para describir máquinas de Turing, es fácil demostar que ese número finito es *1*.

Identificar la presencia de estados de tipo variable en una máquina como garantía de que la rutina que transita por cada uno de estos no codifica información numérica explícita resulta tentador en la misma medida que lo fue considerar en su momento repeticiones de instrucciones. En este caso, se aprecia, además, cómo la clasificación de estados propuesta permite resolver el problema que llevó a suspender el estudio del criterio expuesto en [11]:

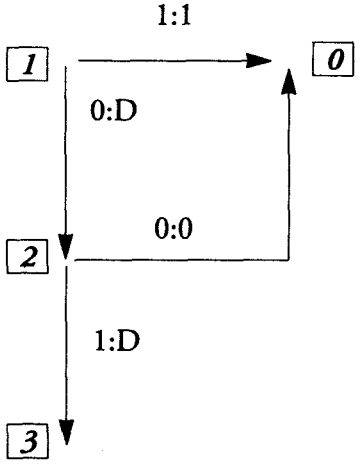
[13] i.

```
10D1
1112
```



ii.

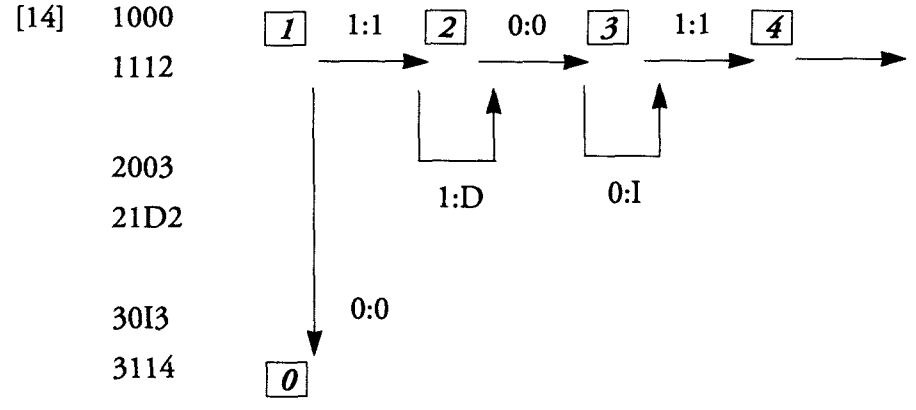
```
10D2
1110

2000
21D3
```



En [13.i] el tránsito entre el estado *1* y el *2* sólo se activa cuando tras una evolución de longitud variable sobre la cinta de cálculo —hacia la derecha de la posición inicialmente ocupada por la cabeza lectora— se detecta finalmente la subsecuencia «01». [13.ii] requiere igualmente la presencia de la subsecuencia

mencionada para activar el estado *3* hallándose activo el estado *1*, sin embargo, es evidente que mientras en [13.ii] esa condición parece asociada a un proceso de codificación explícita de información, en [13.i] tiene lugar como identificación del inicio de una secuencia no vacía de celdas.

La definición de estado de tipo variable no es, por desgracia, puramente sintáctica. En otras palabras, el patrón descrito en [12] puede ser utilizado para caracterizar estados discretos capaces de introducir información numérica explícita. Así,



El estado *3*, pese a constituir una instancia genuina del patrón ofrecido en [12] se comporta, de hecho, como un estado discreto. Obsérvese que cuando la máquina inicia su rutina en *1*, ésta procede a identificar en primer lugar el contenido de la celda sobre la que se halla la cabeza lectora. Si ésta se se encuentra vacía, la máquina se detiene. En caso contrario se activa el estado *2*. Este último puede dar lugar a una evolución de longitud variable aunque finita —suponemos que nuestras cintas contienen a lo sumo un número finito de celdas ocupadas— que se suspende en el momento que en es localizada la primera celda vacía por la derecha. Aisladamente, el estado *3* ejecutaría un recorrido variable sobre la cinta destinado a localizar la primera celda ocupada por la izquierda. Sin embargo, es obvio que su concatenación con el estado *2*, bajo las condiciones inducidas por *1*, hace de *3* un estado discreto: se limita a detener la cabeza lectora sobre la última celda ocupada.

La operación de estados discretos en la rutina de una máquina se reconoce con frecuencia en la identificación y posible actuación sobre ciertas celdas de algún modo significativas. Ejemplos de este tipo de celdas especialmente relevantes son la primera o la última celdas de una secuencia de celdas ocupadas, la celda que separa dos secuencias ocupadas, etc. La función de estos estados no se agota sin embargo, en el tipo de tarea descrita. Como ya se ha dicho, son igualmente capaces de operar dando lugar a un proceso de codificación de información numérica explícita. Lo que separa una y otra posibilidad parece ser el contexto en el cual se produce la activación de un estado de tipo discreto. Si ésta tiene lugar tras la operación de un estado de tipo variable y da paso a otro del mismo tipo, su función deberá limitarse, esta es mi hipótesis, a la identificación de alguna celda notable referida a las secuencias de celdas entre las que se ubica —identificadas o recorridas por los estados variables mencionados. Si, por el contario, un estado discreto da paso a la activación de otro también discreto, la imposibilidad de que ninguno de estos identifique celdas relevantes en secuencias de longitud variable se resuelve en la codificación de información numérica explícita. La máquina que borra la última celda de una secuencia no vacía de celdas ocupadas o aquella otra que procede a rellenar la celda que separa dos secuencias no vacías de celdas ocupadas son ejemplos de máquinas en las que la operación de un estado discreto queda acotada por la operación de estados variables. La máquina que procede a borrar las dos últimas celdas de una secuencia de celdas ocupadas es una instancia de una máquina en la que la concatenación de dos estados discretos sirve al efecto de codificar información numérica explícita.

El curso de esta discusión indica la necesidad de contar con ciertas definiciones auxiliares.

Definición 3. Dada una máquina M cuya tabla consta de *m* estados, diremos que el estado *n'* es *1-activable* desde *n* —$n \leq_1 n'$ en símbolos— si es posible mostrar la existencia de una configuración admisible en la que sólo ha de escrutar una celda antes de dar paso a la activación de *n'*.

En general, *m* es *k-activable* desde *n*, $n \leq_k m$, si existe una configuración admisible en la que el número de celdas que la máquina ha de escrutar para activar *m* hallándose activo *n* es *k* —no importa que para ello deban activarse otros estados intermedios. Diremos finalmente que *m* es activable desde *n*, $n \leq m$, syss existe un entero positivo *k* tal que $n \leq_k m$.

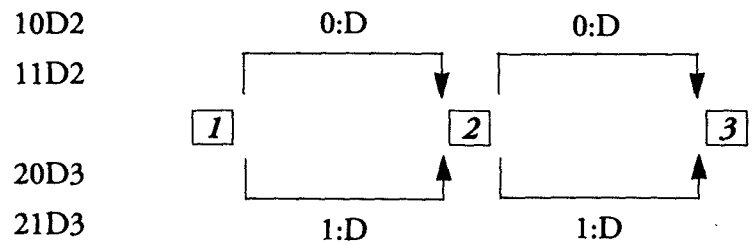
Lo que líneas atrás se denominaba estado variable puede reinterpretarse ahora como aquel para el que se cumple que $n \leq_k n$, $\forall k$, existiendo además un estado *m* y un entero positivo *j* tal que $n \leq_j m$, siendo $n \leq m$. Los estados discretos satisfacen por su parte el requisito de que $\forall m$, si $n \leq_1 m$, entonces $n \leq m$[8].

Estas nociones auxiliares permiten interpretar los comentarios anteriores en un sentido algo más riguroso. En particular, lo que parece desprenderse de ellos es la conexión entre la presencia de información numérica explícita en la subrutina que media entre un estado *n* y otro *m* y la posibilidad de probar el siguiente aserto:

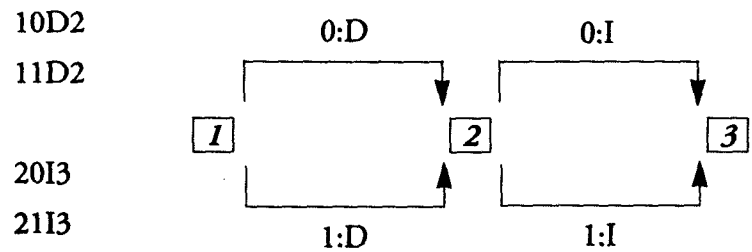
[15] Si $n \leq m$, entonces si $\forall k,j$, tales que $n \leq_k m$ y $n \leq_j m$, sucede que $k=j$.

Esta afirmación, empleada como criterio, permite localizar multitud de procesos en los que estamos dispuestos a admitir la presencia de información numérica explícita. Así, por ejemplo, si se tiene en cuenta que la definición de *k-activabilidad* no indica nada acerca de si las celdas escrutadas han de ser nuevas o pueden ser celdas ya analizadas, las máquinas,

[16] i.



ii.



8 Un estado discreto que se halla activo puede volver a activarse al final de la ejecución de una cierta subrutina, pero nunca lo hará es un solo paso, esto es, escrutando una sola celda.

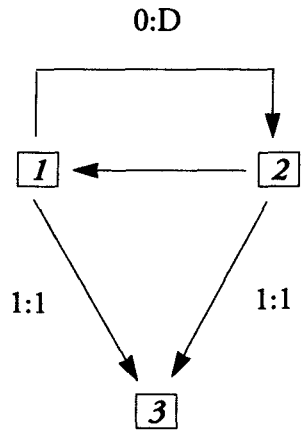
constituyen ejemplos en los que las subrutinas que median entre *1* y *3* incorporan en ambos casos información numérica explícita. Es tentador considerar, además, que la información numérica codificada es, precisamente, la que establece el entero positivo mencionado en [15] representado por *k*.

No obstante, es posible definir máquinas en las que la forma de codificar información explícita adopta expresiones algo más complejas. La siguiente es un ejemplo sobre el que conviene detenerse:

[17]

10D2
1113

20D1
2113



Es evidente que $1 \leq 1$ y lo es, además, del mismo modo en que resulta obvio que $1 \leq_{2n} 1$, $\forall n>1$. Esta observación obligaría a aceptar que las subrutinas que tienen lugar entre la activación de *1* y cualquier otra activación ulterior de ese mismo estado no codifican, al incumplir [15], información numérica explícita: no existe un único entero positivo que indique el número de celdas que es preciso escrutar para activar *1* hallándose activo ese mismo estado. Es difícil admitir esta consecuencia una vez se ha constatado el carácter aparentemente explícito de la condición que debe cumplirse para que *1* vuelva a ser activado. Más bien parece que lo que se impone es ampliar el alcance de [15] incluyendo cualquier tipo de periodicidad en la activación de un estado que sea expresable por medio de algún patrón numérico explícito. Esto nos lleva a proponer el siguiente criterio:

[18] Si sucede que $n \leq_k m$ para algún $k \leq 2$, entonces $n \leq_s m$ $\forall s$, $s \leq 2$.

Puesto que en el caso de [17] *1* sólo vuelve a ser activado cuando se han escrutado al menos dos celdas, el antecedente de [18] resulta inmediatamente satisfecho, aunque ya no lo es su consecuente. Utilizar [18] como criterio para evaluar la codificación de información numérica explícita en las subrutinas que trascurren entre dos estados *n* y *m* parece pues bastante prometedor. Su uso sobre [17] permite, además, apreciar fenómenos de cierto interés.

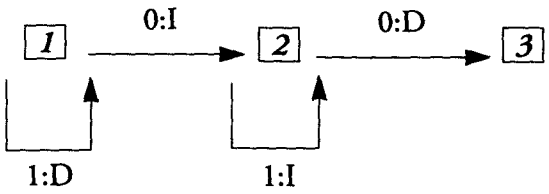
Del mismo modo que se comprueba que $1\leq_{2n}1$, $\forall n>1$, se puede demostrar que $1\leq_{2n+1}2$, $\forall n>0$. Si ahora se evalúa el modo en que *3* puede ser activado desde *1* se observa, sin embargo, que la combinación de las condiciones anteriores da lugar a que $1\leq_k 3$, $\forall k$. La existencia de máquinas como la que se define en [17] descubre problemas de una dimensión difícil de evaluar en este momento. Nos enfrenta a una decisión en cierto modo inesperada: ¿qué debemos decir de máquinas que contengan subrutinas como la que se ejecuta al pasar de *1* a *3*?, ¿codifican información arbitraria o no lo hacen?

Para proseguir con estas y otras cuestiones hace falta, no obstante, cerciorarse de la presunta bondad de [18].

[19]

10I2
11D1

20D3
21I2



Aunque de apariencia inocente [19], demuestra cómo es posible programar recorridos sobre la cinta de cálculo en los que el número de celdas, pese a variar de unos insumos a otros, sigue un patrón constante que indica la presencia de información numérica explícita. Es evidente que $1\leq 3$ y además, se puede afirmar igualmente que existe un insumo para el cual $1\leq_k 3$ para $k\geq 2$. Sin embargo, se aprecia de inmediato que lo que realmente sucede es que $1\leq_{2k}3$, $\forall k$, resultando así el consecuente de [18] insatisfecho. Esta máquina muestra cómo nuestro criterio es sensible a recorridos de tipo periódico no sólo tomando una celda ocupada como unidad —tal y como hace [17.ii], sino también tomando una colección de longitud variable de celdas ocupadas como unidad de medida del período.

Esta última idea se muestra de nuevo en consonancia con el criterio considerado en [15] y según el cual, lo que se trataría de impedir es la repetición de un cierto modelo o patrón en el proceso de escrutinio de la cinta de cálculo durante la ejecución de una determinada subrutina. [18], aunque de manera sutil, se aparta de esta motivación para plantear algo que, bajo ciertas condiciones, puede resultar excesivamente restrictivo. La conexión entre un estado inicial *1* y otro final *0* puede, por el mero hecho de mostrar una relación funcional entre insumos y exsumos, comprometer severamente la aplicabilidad de [18]. Sin embargo, nunca debe ser ésta la razón que nos impulse a identificar una máquina como portadora de información numérica explícita, sino, más bien, el modo en que se llega a establecer la conexión entre los valores de los insumos y los posibles valores de los exsumos. Es preciso reconocer, que [15] se muestra en esto mucho más razonable que [18].

Cuarto modelo

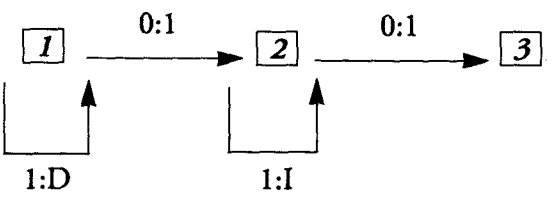
Para reconocer si un patrón de lectura de una cinta de cálculo se repite a lo largo de una cierta subrutina basta, en realidad, tomar dos secuencias acotadas por los estados correspondientes. Esto hace preciso considerar un máximo de cuatro estados, n_i, n_j, n_k y n_m con el fin de identificar las secuencias aludidas, estados que, claro está, pueden verse eventualmente reducidos a tres si identificamos, por ejemplo, n_j y n_k. A continuación, habrá que analizar qué sucede para que n_j se active cuando la máquina ejecuta las instrucciones correspondientes a n_i. Una vez satisfecho este trámite, deberá estudiarse esa misma cuestión referida esta vez a los estados n_k y n_m. Si se demuestra que cualquier configuración admisible de la máquina que se analiza se comporta de manera que el número de celdas requeridas para activar n_j desde n_i resulta siempre el mismo que se precisa para activar n_m desde n_k, entonces y sólo entonces resultará necesario admitir la repetición de un patrón fijo como el que se ha mencionado líneas atrás.

Es importante entender que la identificación de un patrón de lectura cuya repetición se observa a lo largo de una subrutina no tiene por qué producirse de forma consecutiva. En [19], por ejemplo, es inmediato comprobar cómo el patrón de lectura se reproduce sin solución de continuidad, pero nada hubiera impedido intercalar estados innocuos entre los recorridos mencionados, o incluso, subrutinas de cierta complejidad. Una observación mucho menos evidente que ésta es la que aconseja reemplazar la mera repetición del número de

celdas escrutadas por un criterio más elaborado. En realidad uno que incorporase lo que en muchas ocasiones sólo parecen convenciones incorporadas en la sintaxis con que se definen las máquinas de Turing.

[20] 1012
11D1

2013
21I2



La tabla de [20] ha sido elaborada a partir de una mínima modificación sobre la tabla de [19]. No obstante [20], no repite un patrón en el mismo sentido que [19]: en *1* recorre la cinta hacia la derecha hasta encontrar la primera celda vacía la cual procede a rellenar. A continuación *repite* el proceso sólo que en sentido inverso rellenado igualmente la primera celda vacía por la izquierda. Mientras que en el recorrido inicial —hacia la derecha— escruta *n* celdas, el recorrido inverso —hacia la izquierda— siempre escruta n+1 celdas. ¿Se trata éste de un caso de codificación de información numérica explícita? Por el momento, me limitaré a considerar esa opción en el criterio que se ofrezca más adelante.

Para traducir estas intuiciones informales en un criterio riguroso se precisa retomar la idea original relativa a la dicotomía establecida entre estados discretos y de tipo variable. Lo que ahora interesa es si un estado es capaz de permanecer activado más de una celda antes de provocar la activación de otro posterior, o si es capaz, por el contrario, de permanecer activado un número variable aunque finito de ellas. Aunque la nueva noción puede ser recuperada desde la de k-activabilidad, ahorraré explicaciones innecesarias a lector si la presento como primitiva[9].

[9] La única diferencia entre $\leq k$ y $\Rightarrow k$ reside en que al afirmar n(km se está admitiendo la existencia de una subrutina entre la activación de n y la de m, mientras que al afirmar $n \Rightarrow km$ entre la activación de n y la de m no puede suceder ningún otro evento.

Definición 4: Dados dos estados *n* y *m* pertenecientes a una máquina de Turing M,

i. $n \Rightarrow_0 m$ syss para toda posible configuración de M sucede que *n* sólo es capaz de permanecer activada el período correspondiente al único escrutinio de una celda antes de dar paso a la activación de *m*.

ii. $n \Rightarrow_k m$, $k \neq 0$, syss existe una configuración admisible de M en la que *n* permanece activado el período correspondiente al escrutinio de k celdas antes de dar paso a la activación de *m*.

El criterio que mejor reúne los comentarios precedentes no puede ser ahora más inmediato.

[21] Dados cualesquiera dos períodos acotados por los correspondientes estados n_i, n_j, n_k y n_m en la tabla de una máquina de Turing diremos que la subrutina que transita por n_i, n_j, n_k, n_m codifica *información numérica explícita* si es posible mostrar que para toda configuración admisible de M sucede que:

— Si $n_i \Rightarrow_x n_j$ y $n_k \Rightarrow_y n_m$, entonces $| x-y | \leq 1$.

La versión más permisiva de [21] se limitaría a considerar aquellos casos en los que x=y aceptando entonces [20] como ejemplo de rutina libre de información numérica explícita. Queda para el lector comprobar cómo [21] continúa dando satisfacción a todos los ejemplos discutidos hasta el momento en este ensayo.

5. Información numérica explícita: criterio y consecuencias

Aunque debo reconocer que las discusiones detalladas y prolijas como la anterior no son mi modelo de exposición, no veo de qué forma se hubiera podido evitar incurrir en esos defectos —si realmente lo son— sin afrontar otro mal aún mayor: el de la discusión puramente técnica comprensible tan sólo para el afanado autor.

Finalizaré, pues, indicando tan sólo las principales decisiones alcanzadas, evaluando algunas de las consecuencias que éstas imponen sobre la investigación formal subsiguiente.

Aunque [21] parece dar respuesta a muchas de las demandas intuitivas que aquí se han discutido, por sí solo no es capaz de establecer cuándo una máquina codifica información numérica explícita. Para ello es necesario poner su contenido en relación con la rutina que ejecuta una máquina. Lo más directo resulta, a buen seguro, decir que

[22] Una máquina *M codifica información numérica explícita* si es posible localizar dos subrutinas acotadas por estados n_i, n_j, n_k y n_m en su tabla que satisfacen de manera no trivial [21].

La conexión con la noción de conjunto definible en sentido constructivo o como sostuve en [6], dotado de una definición constructivamente aceptable, se efectúa ahora en dos pasos.

[23] Un conjunto numérico (tiene una *construcción perfecta* syss la función $\varphi(x)$ en $\Gamma=\{x / \varphi(x)=0\}$

es computada por una máquina libre de información numérica explícita.

No deja de resultar una ironía que la definición anterior, presuntamente imbuida del espíritu de la matemática constructiva, opere con un criterio [21], el cual no resulta efectivo sobre la clase de entidades de las cuales se predica. No obstante, esto es algo que parece inherente a cualquier solución constructivamente aceptable y ante lo cual lo único que cabe hacer es definir mecanismos destinados a diseñar de máquinas consistentes con [21].

Conjuntos para los cuales se pueda ofrecer una definición actual del tipo de la de [23] no abundan ciertamente. El conjunto (de los números naturales es uno de ellos y aunque la lista no es fácil de extender, aún me atrevería a mencionar, por ejemplo, el de los números primos. Si se extiende [23] aceptando la definición de subconjuntos en $\mathbb{N}^n$ por medio de funciones n-arias la lista au-

menta de forma significativa. El conjunto formado por las tuplas <x,y,z> tales que x+y=z, o tales que x·y=z son un buen ejemplo de ello.

Una vuelta al espíritu de mis primeras consideraciones obliga a reconocer que tal vez el intento de aislar el uso de información numérica explícita ha ido demasiado lejos en un cierto sentido. La definición del conjunto de los enteros positivos *pares* resulta, entendido en sus justos términos, un ejemplo de construcción legítima en la que, no obstante, hay mención explícita a cierta información numérica —el *2* en este caso. Es evidente, aunque no se muestre ahora, que toda máquina capaz de computar la función característica unitaria $\chi(x)$ de ese conjunto debe codificar según [22] información numérica explícita. No obstante, nada impide que en lugar de considerar la función $\chi(x)$ evaluemos la construcción del conjunto de los enteros pares como una instancia particular —que es lo que en definitiva representa— de la función característica $\chi(x,y)$ que dirime si x es divisible por y. La forma en que es posible expresar el carácter constructivo del mencionado conjunto consiste ahora en analizar la ausencia de información numérica explícita en la máquina que computa la función $\chi(x,2)$. Al situar la presencia del entero positivo responsable del conflicto como argumento de una función computable, su impacto sobre la tabla de la máquina involucrada desaparece. Esto permite introducir la siguiente definición:

[24] Un conjunto numérico (tiene una *construcción aceptable* si la función (n+1) aria $\varphi(x,k_0,...k_n)$ que figura en

$$\Gamma=\{x \,/\, \varphi(x,k_0,...k_n)=0\}$$

es computada por una máquina libre de información numérica explícita y además,

$$\{k_0,...k_n\} \neq \Gamma \quad y \quad \{k_0,...k_n\} \neq \mathbb{N}\text{-}\Gamma.$$

En principio, y a falta de una discusión más pormenorizada, ningún conjunto arbitrario finito tiene una construcción aceptable en el sentido de [24]. No obstante, dado un conjunto de estas características, aquel que se obtiene, por ejemplo, al considerar todos los múltiplos de alguno de los elementos de

dicho conjunto sí gozaría del calificativo introducido en [24]. La razón es simple: igual de arbitraria es la elección del entero positivo *2* a la hora de definir el conjunto de los números pares que cualquier otra sobre la cual se opere de alguna forma para obtener otro conjunto a continuación.

Una pregunta en cierto modo inevitable es aquella que plantea la hipotética eliminabilidad de toda información numérica explícita a través de expedientes como el que se introduce en [24]. Una vez identificada información numérica en la tabla de una máquina, siempre se podrá construir otra capaz de interpretar la información numérica presente en términos de valores paramétricos en los argumentos de una función computable libre de información numérica explícita. Esta hipótesis no debería confundirse nunca con el contenido del archiconocido *teorema de los parámetros* -(s-m-n)-Theorem, cuya presencia establece, más bien, la computabilidad de aquellas funciones obtenidas de otras también computables mediante el expediente de fijar ciertos argumentos como parámetros. Es evidente que la clase de las funciones computables es, en este sentido, cerrada bajo el alcance del Teorema de los parámetros. Su aplicación, es, sin embargo, el tipo de expediente que estamos interesados en excluir a la hora de definir la clase de todas aquellas máquinas libres de información numérica. En este sentido se puede afirmar que el Teorema de los parámetros constituye la contrapartida de la hipótesis que acabo de plantear y cuya formulación rigurosa sería:

[25] Sea $\varphi(x_1,...x_n)$ la función que computa una máquina M^{φ} la cual incluye en su tabla información numérica explícita en $\{k_1,...k_n\}$. Existe entonces una función numérica $\chi(x_1,...x_n,k_1,...k_n)$ computada por una máquina M^{χ} la cual no codifica información numérica en $\{k_1,...k_n\}$ en su tabla.

Ciertamente, sería una sorpresa localizar alguna máquina para la cual fuese imposible verificar [25], no obstante, por muy obvio que me pueda parecer la veracidad de este aserto, no me atrevo en estos momentos a postular sin más su aceptabilidad: hay un cierto sentido en el cual me parece que un resultado como este requiere demostración.

Bibliografía

GOODMAN, N., «Intensions, Church's Thesis, and the Formalizations of Mathematics», *Notre Dame Journal of Formal Logic*, vol. 28, n.° 4, 1987, pp. 473-489.

HEYTING, A., «Constructivity in Mathematics», *Proceedings of Colloquium, Amsterdam, 1957*, North-Holland, Amsterdam, 1959.

KLEENE, S., «Reflections on Church's Thesis», *Notre Dame Journal of Formal Logic*, vol. 28, n.° 4, 1987, pp. 490-498.

LIFSCHITZ, V., «Constructive Assertions in an Extension of Classical Mathematics», *The Journal of Symbolic Logic*, vol. 47, n.° 2, 1982, pp. 359-387.

SOARE, R., «Computability and Recursion», *The Bulletin of Symbolic Logic*, vol. 2, n.° 3, 1996.