Efectividad computacional y máquinas

Javier Blanco; Pío García
Universidad Nacional de Córdoba

I.Introducción:

Para demostrar la solución negativa al *Entscheidungsproblem*, Turing caracterizó formalmente la idea de efectividad. Para ello, redujo los procedimientos a un conjunto pequeño de pasos elementales a ser realizados por un computador humano. Esta noción era coincidente con las propuestas de Church. No obstante, la caracterización de Turing fue considerada como una "solución" al problema de especificar en qué consistía un procedimiento efectivo. Es más, Gödel consideró que Turing nos había dado "una definición absoluta de una noción epistemológica interesante" (Gödel 1990, p. 150).

Es una cuestión controvertida explicitar las razones por las cuales la comunidad que estaba trabajando en la noción de procedimiento efectivo consideró a la presentación de Turing como una solución. Más allá de las *razones históricas* particulares de este acuerdo generalizado, hay una cuestión *conceptual* más amplia que involucra el alcance y significado de la solución sugerida por Turing. Lo cierto es que la convergencia de las propuestas de procedimiento efectivo favoreció una caracterización *extensional* de aquello que es computable. En este trabajo avanzaremos en una caracterización más de tipo intensional de la computación. Para ello distinguiremos, en primer lugar, entre efectividad y programabilidad, como dos aspectos de la computación. En segundo lugar sugeriremos dos maneras de caracterizar las nociones de *efectividad* y *programabilidad*, una de ellas focalizada en el análisis de las *intuiciones* supuestas y otra centrada en la *axiomatización* de propiedades relevantes. Estimamos que este tipo de análisis puede ser ventajoso para abordar diversos problemas en filosofía de la computación. En concreto, la hipótesis que desarrollamos aquí, es la necesidad de considerar por separado dos conceptos diferentes, que se aplican en dominios diferentes, y cuya intersección da lugar a la idea de computación efectiva y a la tesis de Church-Turing. Por un lado, la idea de *efectividad* que parece, en principio, asentarse en la de mecanismo. Por otro, una idea relacional de "computación" – especificada en términos de programabilidad- . Es decir, desde esta perspectiva "computar" involucra un tipo de relación que puede tomar distintas formas. Así, por ejemplo, se podrían citar las relaciones entre codificaciones de comportamientos y los comportamientos mismos, entre programas como scripts -descripciones- y programas como procesos. Dichas relaciones pueden estar asociadas a funciones sub-recursivas, recursivas o super-recursivas (Burgin), siendo la uniformidad de estas relaciones algo a considerar. Esta manera de presentar la computación supone una consecuencia habitualmente aceptada, que hay procedimientos que tienen la estructura relacional de la computación pero que van más allá de lo computable – hipercomputación-, y una consecuencia hasta cierto punto controversial, que podría haber efectividad más allá de los procedimientos relacionales computables. Se puede plantear esta cuestión sugiriendo que hay una noción *intuitiva* de efectividad asociada a las máquinas – a partir de la estructura funcional de las mismas o de otra propiedad- y una noción de

efectividad asociada con procedimientos – en la tradición del cálculo. Es el encuentro de estas dos intuiciones lo que parece explotar Turing.
Organizaremos nuestro trabajo de la siguiente manera. En la segunda sección presentaremos las ideas básicas de nuestro análisis de la noción de efectividad. En la tercera sección explicitaremos las razones por las cuales estimamos que la noción de programabilidad es central para entender a la computación. En la cuarta sección sugeriremos la axiomatización de propiedades relevantes como una manera de lograr una caracterización intensional no sólo de la computación sino de las nociones básicas supuestas. En las consideraciones finales presentaremos algunas de las ventajas que creemos tiene esta manera de entender la computación.

II. Caracterizando la efectividad

En la presentación del parágrafo 9 de su artículo de 1936, Turing vincula a través de un par de recursos el cálculo matemático con las máquinas. El primer recurso que introduce es la consideración del procedimiento que lleva adelante un ser humano para solucionar un problema de cálculo. El segundo recurso es la cualificación de este procedimiento como "mecánico". Como decíamos en la introducción, Gödel calificó al concepto de computación de Turing como una noción epistémica "absoluta". De acuerdo con Kennedy (2014) esta apreciación de Gödel estaba asociada con el carácter "intuitivo" y no circular de esta noción. No parece problemático decir que gran parte de la fuerza de la intuición provenía de lo que suponemos que puede hacer una máquina. Menos directa es la manera particular en la cual se pueden analizar estas intuiciones. En esta sección sugeriremos, primero, algunas hipótesis que servirán para desarrollar estas intuiciones, y luego presentaremos alguna evidencia para sostener la plausibilidad de estas hipótesis.
Una manera de elucidar la relación entre efectividad y una noción relacional de computación es a través del análisis que hace Copeland, en el contexto de la evaluación del alcance de la tesis Church-Turing, de lo que llama la tesis M: "Whatever can be calculated by a machine (working on finite data in accordance with a finite program of instructions) is Turing-machine-computable." (Copeland 2002). Habría, dice Copeland, dos interpretaciones posibles de la tesis M, dependiendo del tipo de máquina considerado. Así, en una primera interpretación la única restricción a considerar serían las físicas – que se ajuste a las leyes que conocemos del mundo-. En una segunda interpretación de la tesis M no habría restricciones físicas a considerar sino sólo las abstractas – sólo se tomaría en cuenta una máquina nocional-. Las investigaciones sobre hipercomputación implican que, bajo la segunda interpretación, la tesis Church- Turing es falsa. Es una cuestión a debatir si lo mismo ocurre con la primera interpretación.
Si, como decíamos más arriba, se podría distinguir entre un ámbito de procesos efectivos y un ámbito de procedimientos relacionales cuya intersección constituiría la computación, entonces se debería poder caracterizar a los procesos efectivos independientemente de los computacionales. Si bien esta tarea no parece sencilla ni directa, por la historia de la elucidación de lo que es la computación, la referencia a las máquinas que aparece en el parágrafo 9 del artículo clásico de Turing habilita y sustenta una indagación de este tipo.
Una estrategia habitual entre aquellos que intentan dar cuenta de las máquinas es distinguir entre un mecanismo y la organización en la cual dicho mecanismo se inscribe (cfr. Canguilhem 1952 y Piccinini) 5. Un mecanismo – y el "ensamble"6 que constituye- puede ser entendido, desde una perspectiva de diseño o funcional, como un procedimiento. Si este procedimiento tiene las propiedades adecuadas – entre las cuales se destaca la finitud- entonces puede servir para entender a los procedimientos efectivos -computacionales-. De esta forma lo presentó Turing. La finitud

estaría vinculado con el número de pasos de instrucciones de un procedimiento y con la cantidad de pasos que debería dar para llegar a una solución (Cfr. Copeland 2002).
Pero una máquina, en principio, podría llevar adelante sus objetivos, llegar a un resultado, con mecanismos de los cuales no conocemos si tienen las propiedades antes señaladas. Este breve panorama podría ser entendido de la siguiente manera. La intuición acerca de la efectividad, en este tipo de contextos, parece estar asociada con las máquinas. Las máquinas pueden ser efectivas a través de procedimientos de tal manera que sean individualizadas como computacionales – si tienen las propiedades adecuadas-. Pero las máquinas pueden ser efectivas si logran un resultado de acuerdo con su diseño, aunque no sepamos si sus mecanismos satisfacen propiedades como las señaladas más arriba. A los fines de lograr mayor claridad llamemos *procedimientos* a los primeros y *mecanismos* – sin más- a los segundos 7. Los procedimientos son efectivos si tienen las propiedades adecuadas 8, los mecanismos son efectivos si cumplen su función dentro del ensamble que constituye la máquina.
Hay varias objeciones a esta propuesta de analizar el significado intuitivo de la noción de efectividad. Según Copeland "'Effective' and its synonym 'mechanical' are terms of art in these disciplines: they do not carry their everyday meaning.". De esta manera parece que, en principio, indagar el sentido de mecánico -como procedimiento efectivo- a partir de las máquinas es una estrategia destinada al fracaso. Además, de acuerdo con este autor, el carácter *intuitivo* de la noción de procedimiento efectivo desaparecería – se reemplaza- por la caracterización *formal* de máquina de Turing. El alcance de esta última apreciación es habitualmente entendido como una "hipótesis de trabajo" - tal como el propio Copeland destaca -, por lo tanto difícilmente pueda desaparecer el sentido intuitivo de efectividad.
Más complicado de contestar es la primera objeción. Como decíamos en la introducción hay un costado histórico en este problema que no parece sencillo aclarar con la información disponible. Pero, hay una cuestión conceptual, no obstante, en la cual se puede utilizar evidencia de otro tipo. La vinculación con las máquinas – computadoras como artefactos- ya aparece con claridad en el siglo XIX. Sin embargo, la caracterización que hemos esbozado más arriba supone que la idea de efectividad puede bifurcarse a partir de las nociones de procedimiento y mecanismo. Parece evidente que ambas nociones estaban vinculadas en el nacimiento de la computación. No es claro la forma en la cual se constituye este vínculo.
A los fines de nuestro trabajo es suficiente señalar que hay un vínculo íntimo entre ambas nociones. Sin embargo, podríamos especular, de manera tentativa, acerca de la posible naturaleza de la vinculación entre ambas nociones. En la propuesta de 1936, Turing construye sus máquinas automáticas a partir de la consideración de los procedimientos mecánicos realizados por un ser humano. La intuición parece asentarse en el funcionamiento de las máquinas. Pero estas se construyen a partir del análisis de los procedimientos mecánicos. Procedimientos mecánicos y mecanismos de máquina podrán distinguirse más adelante de manera clara, pero en el nacimiento de la computación estas nociones parecen apoyarse mutuamente. Si esta sugerencia altamente especulativa es plausible, entonces la intuición de la caracterización de efectividad parece sustentarse en la vinculación – de doble vía- entre procedimientos y mecanismos. No obstante, como ya lo dijimos más arriba, nuestra propuesta no depende de la plausibilidad de esta hipótesis.

III. Programabilidad

¿Cómo caracterizar el aspecto relacional de la computación que señalábamos más arriba? Los sistemas que llamamos computacionales tienen como propiedad identificatoria la programabilidad. Las computadoras son artefactos cuyo comportamiento es modificable por instrucciones de carácter prescriptivo. Veamos brevemente qué entendemos aquí por programabilidad.
En trabajos anteriores, en colaboración con Renato Cherini, hemos propuesto la centralidad de la noción de programabilidad para comprender a los sistemas computacional por medio de una caracterización de lo que hemos llamado "intérprete". La idea central que presentamos es que un intérprete produce comportamientos a partir de una codificación de los mismos que acepta como entrada. La relación entre tal entrada, que llamamos programa, y el comportamiento que prescribe, se caracteriza mediante una función de interpretación.

Definición 1 (Función de intepretación). Dado un conjunto B de posibles comportamientos, un conjunto P de programas, y un conjunto D de datos, una función de interpretación es una función $i \in P \rightarrow (D \rightarrow B)$, que asocia a todo comportamiento $b \in B$ un programa $p \in P$ y un dato $d \in D$. Con $A \rightarrow B$ denotamos el conjunto de funciones totales de A en B.

Usualmente referimos a una función de interpretación $i \in P \rightarrow B$, asumiendo que los datos de entrada se encuentran codificados con el mismo programa. Cuando los elementos de P se construyen utilizando algún lenguaje, lo llamamos lenguaje de programación. Cuando consideramos los comportamientos de salida de datos, suele darse que $B = D$.
Las nociones de función de interpretación y programa son relacionales. Una función de interpretación es tal cuando prescribe comportamientos a un conjunto dado de programas; es una función de interpretación para tal conjunto. Un programa es tal cuando existe una función de interpretación para él. No hay nada intrínseco en ser un programa más alla de la relación con una función de interpretación. Por ejemplo, los números de Gödel permiten considerar los números como programas (Davis y Weyuker, 1983). De esta manera, los conceptos de función de interpretación, programa (e incluso lenguaje de programación) son interdefinibles.
Decimos que un sistema es programable en la medida en que admita la codificación de una variedad de comportamientos. El grado de programabilidad del sistema está dado por la variedad de comportamientos que el lenguaje de programación subyacente es capaz de codificar, y es la característica distintiva del sistema computacional como tal. Si consideramos que un sistema es computacional cuando es programable, entonces "ser computacional" es una propiedad que puede ser establecida sólo en relación a un conjunto de comportamientos y una codificación correspondiente. En otras palabras, la propiedad de ser computacional no tiene sentido independientemente de un conjunto de comportamientos y un conjunto de programas.

IV.Axiomatización

Como decíamos en la introducción, habría una segunda manera de caracterizar la distinción entre efectividad y programabilidad. Axiomatizar la noción de computación efectiva tendría una consecuencia teórica de gran importancia, ya que permitiría "verificar" la tesis de Church-Turing, es decir, mostrar que podría demostrarse como consecuencia de los axiomas propuestos. En el ámbito filosófico, una axiomatización permitiría determinar cuáles serían los modelos que satisfarían dichos axiomas y por lo tanto se podría tomar ese camino para la difícil tarea de caracterizar cuándo un sistema computa. Los intentos de axiomatización se han enfocado en alguno de los dos aspectos considerados más arriba (la efectividad o la codificación), dejando de hecho al otro como subsidiario de este. Revisaremos algunos ejemplos en esta clave, como los de

Gandy, Sieg, Dershowitz y Gurevich, que buscan caracterizar la idea de efectividad, es decir, de cálculo mecánico; o los de Wagner, Strong, Fenstad, Moschovakis que intentan dar cuenta de la relación entre programas (o índices) y comportamientos (en general, funciones).

Las axiomatizaciones de esta noción suelen basarse en el análisis de Turing acerca de la computación humana, y el posterior análisis de Gandy de los mecanismos. Los axiomas presentados intentan dar cuenta de las cuestiones estructurales y comportamentales de los mecanismos y la relación entre ambas. Varios trabajos interesantes pueden ser considerados en esta línea (Turing, Gandy, Sieg, Copeland y Shagrir, Dowek, Dershowitz y Gurevich).

Por un lado, se da cuenta de que un proceso mecánico se lleva adelante paso a paso. Aparece acá la noción informal de *algoritmo*, como una descripción de un proceso reducido a pasos elementales. Suele identificarse aquí al algoritmo mismo con el proceso al considerar a este como un sistema de *transición de estados.*

Tanto si se consideran humanos como máquinas, ciertas cotas acerca de estos sistemas tienen que ser postuladas para no trivializar la idea de efectividad, es decir, para que no termine aplicándose a cualquier sistema. Para un humano, las cotas aparecen en las limitaciones del sistema perceptivo, de acción y de memoria (este último admite ciertas discusiones acerca de su finitud); para un mecanismo las restricciones suelen aplicarse a su constitución estructural (número finito de partes, complejidad acotada, etc).

Por último, se postulan restricciones acerca de cómo se producen los pasos de ejecución, requiriendo a veces determinismo (a lo sumo un paso posible en un estado dado) y localidad en el caso de los mecanismos, admitiendo solo un conjunto acotado y cercano de partes como posibles generadores de una acción.

La deficiencia clara de estas axiomatizaciones es que no distinguen entre sistemas computacionales y sistemas mecánicos en general, posibilitando entre otras cosas, por cuestiones de realizabilidad múltiple, que cualquier sistema suficientemente complejo pueda verse como computando cualquier programa, el llamado problema del *pan-computacionalismo.*

Para poder caracterizar la idea de computación a partir de una codificación dada, suele considerarse primero un dominio sobre el cual se definen funciones, y luego una manera de relacionar (algunas) funciones sobre ese dominio con elementos del mismo. Esa relación misma es constitutiva del sistema computacional y las aproximaciones axiomáticas buscan caracterizarla. Usualmente, ya sea como axiomas o como teoremas a partir de ellos, aparecen varias de las propiedades que Kleene estudia de las funciones recursivas en sus varias versiones, como son el teorema de enumeración, el teorema de la recursión, el teorema de punto fijo, la propiedad smn (o el teorema del parámetro), etc.

La relación entre índices y funciones, la cual puede verse como una generalización de la relación entre programas y sus comportamientos de entrada-salida, se pone en foco y se busca expresar axiomáticamente sus propiedades, en particular para ver cómo las operaciones en los índices pueden realizarse de manera uniforme, estableciendo una operatoria abstracta sobre estos que, se espera, capturen lo esencial de la idea de computabilidad. Esta relación se establece de manera prescriptiva, en las llamadas Uniformly reflective structures (URS) aparece como uno de los elementos que caracteriza el sistema formal. Los índices son siempre posibles datos de las funciones.

Consideraremos estas versiones axiomáticas, teniendo en cuenta dos de sus limitaciones esenciales para poder proponer una presentación conceptualmente más adecuada. Por un lado, las propiedades solo valen para los sistemas hoy llamados Turing-complete, es decir, que sean extensionalmente equivalentes a las funciones calculables por máquinas de Turing. Esto deja

afuera sistemas más limitados (algoritmos sub-recursivos, según la nomenclatura de Burgin) y sistemas más potentes (los algoritmos super-recursivos). Consideramos que podrían encontrarse axiomatizaciones para esos sistemas, ya que la noción que comparten todos ellos es la definición relacional del concepto de computación. Por otro lado, tomar solo comportamientos funcionales (relaciones de entrada-salida) es también una limitación histórica que no se condice con el desarrollo de la ciencia computacional. Consideraremos en qué sentidos es posible extender dichas caracterizaciones a formas de comportamiento más amplias.

Parecería que asumiendo algunas propiedades elementales de estos sistemas, en general la existencia de algunas funciones básicas (constantes, proyecciones, composiciones y análisis por casos, por ejemplo), la auto-aplicación de funciones (vía gödelización si es necesario) es suficientemente poderosa para generar todas las funciones recursivas o equivalentemente Turing-computables. Muchos sistemas sub-recursivos tienen políticas de tipado más estrictas, que restringen la aplicación de funciones a datos. Esto podría reflejarse en ciertas restricciones en la función de gödelización, además de la no-existencia de un valor indefinido y de una función indefinida. Es posible establecer enumeraciones recursivas de sistemas de algoritmos sub-recursivos, pero dicha enumeración no va a pertenecer al sistema mismo. Como ejemplo podemos considerar las funciones recursivas primitivas, para las cuales pueden definirse posibles enumeraciones pero que no son ellas mismas recursivas primitivas.

Una manera promisoria de axiomatizar los sistemas computacionales, puede ser partir de la BRFT de Strong, pero debilitando el axioma de enumeración. La propiedad de SMN, o teorema del parámetro o existencia de especialización o evaluación parcial, podría ser otra de las propiedades interesantes a eventualmente debilitar. Valdría, por ejemplo, para las funciones recursivas primitivas.

V.Consideraciones finales

Uno de los problemas complicados y profusamente atacados en filosofía de la computación es el de determinar cuándo un sistema computa, o bien, qué es computación, cómo distinguir entre un comportamiento meramente físico y un comportamiento computacional de un sistema. Distinguir lo mecánico de lo computacional, analizando por separado ambas propiedades puede permitir caracterizar de mejor manera los sistemas computacionales. Curiosamente, parecería que la respuesta puede estar al alcance de la mano, refinando ligeramente algunos de los trabajos existentes.

En el artículo *What is computation?,* Copeland se pregunta cuándo un sistema computa. Distingue correctamente dos componentes de los sistemas computacionales, que llama algoritmo y arquitectura. Construye un sistema SPEC formal que caracteriza a ambas componentes, para recurrir así a las herramientas de la lógica formal. Al estilo Gandy, considera como modelos posibles a entidades dotadas de un sistema de etiquetado (labelling). En base a las especificaciones de la arquitectura y el algoritmo, construye un conjunto de axiomas que caracterizan el comportamiento deseado. Un modelo de esos axiomas construido en términos de la entidad y el etiquetado sería entonces una realización del sistema computacional. Sin embargo, no cualquier modelo serviría, so pena de caer en las trivializaciones de la noción de computación enunciadas entre otros por Putnam y Searle. Lo que Copeland requiere acá es que los modelos sean “honestos”, es decir, que respeten la idea de acción implícita en la especificación SPEC, y que el etiquetado no sea hecho ex post facto. Las condiciones parecen adecuadas, pero no queda claro que sean

suficientes y, sobre todo, es difícil usarlas como criterio de demarcación. Pierden además la potencia que tiene los sistemas formales de ser independientes de los modelos posibles.
Una cuestión importante e implícita en estos trabajos y de alguna manera en el de Gandy, es que el sistema etiquetado en cuestión que conforma el modelo (pero no **es** el modelo, el modelo de SPEC va a ser una construcción del comportamiento de dicho sistema) es un sistema finito. El conjunto de los posibles comportamientos de dicho sistema es sin embargo infinito. La relación entre el sistema y el conjunto de sus comportamientos en cierto sentido replica la noción de "finitismo" de Hilbert, ya que se requiere que cada comportamiento sea mecánicamente generado por un sistema finito. Para un programa dado, una computación particular tendrá, en la construcción de Copeland, un modelo finito (curiosamente quedarían afuera los comportamientos divergentes).
La misma idea puede llevarse a cabo de otra manera que preserve el código mismo en la especificación y que por lo tanto tenga que ser parte del modelo que lo realiza. De hecho, está por ejemplo ya resuelta en la axiomatización de los comportamientos de un lenguaje de programación en un sistema lógico de primer orden dada por Nils Jones en su libro *Computability and Complexity*. Allí, se describen dos predicados que caracterizan de manera completa los comportamientos de un lenguaje de programación elemental llamado *WHILE*. Se definen acá un número infinito pero computable de predicados, inductivamente en la estructura de los programas (de los comandos y de las expresiones en realidad).
Si consideramos un sistema lógico con un predicado R(d, d', d''), definido usando los construidos inductivamente por Jones de la siguiente manera:

$R(d, d', d'') = Prog(d)$ AND $G_d(d',d'')$

El predicado Prog determina si su argumento es un programa correcto en *WHILE*, es decir, es básicamente un parser. No hay problema para definirlo en lógica. El predicado G_d codifica la semántica del programa d como transformador de estados
En lenguaje de Jones los datos son directamente programas, por eso puede usarse directamente el primer parámetro como tal para la definición de G, en otros casos en los cuáles haga falta una codificación, esta puede agregarse en la referencia a G, reemplazando d por la codificación de d en el subíndice.
Ahora, un modelo de R puede construirse a partir de un sistema etiquetado al estilo de Copeland, pero con la diferencia de que tiene que estar representado, como parte de los estados, el programa mismo d. El predicado R está definido para cualquier programa d (para cualquier árbol d, en realidad), por lo cual un modelo debe asociar el comportamiento adecuado a cada programa. Las diferentes semánticas de los lenguajes de programación hacen exactamente eso, lo que sería necesario aquí es incluir la arquitectura propuesta por Copeland como parte del modelo.
Parecería aquí que un modelo de R no sería una computación sino que describiría el conjunto de todas las computaciones posibles en el lenguaje dado. Podría, esperamos, definirse computación al interior del modelo como alguna secuencia maximal de ejecución, hay que ver esto.
La historia misma de la noción de computación pone particular énfasis en la noción de programa. Las máquinas precursoras de las computadoras fueron consideradas tales en tanto disponían de alguna capacidad de codificación y de producir comportamientos a partir de los códigos. Contamos aquí a los telares de Jacquard, a las máquinas de Babbage, todas con la capacidad de responder a codificaciones en tarjetas o cintas perforadas. El motor analítico de Babbage en particular era, aún sin que fuera reconocida como tal entonces, una máquina universal. Sin embargo a partir del

desafío de Hilbert de caracterizar lo efectivo, la idea de programa quedó implícita, y lo investigadores se concentraron en caracterizar lo mecánico. Como extensionalmente coinciden (las funciones calculables por cualquier mecanismo serían las mismas que las computables), la propiedad de que un mecanismo sea programable no tuvo ninguna especificidad relevante. La tesis de Church-Turing terminó de consolidar la concepción extensional de efectividad. La identificación de lo efectivo con lo computable hace perder de vista que lo computable implica propiedades específicas que no todos los mecanismos tienen. Por supuesto que si nos restringimos a las funciones en los naturales, las computables coinciden con las efectivas, pero cuando consideramos mecanismos cualesquiera, por definición efectivos, no todos serán computacionales, no todos serán programables.

Incluso en el trabajo en ciencias de la computación una de las técnicas para analizar y verificar programas más usuales consiste en construir un modelo a partir del programa (o de un programa junto con su contexto de ejecución), y trabajar luego con el modelo, ejemplarmente un sistema de transición o un grafo de trazas, o variantes de eso. El programa mismo desaparece en el análisis. Esta aproximación aparece también en los trabajos fundacionales, por ejemplo así se construye la axiomatización de Dershowitz y Gurevich de lo abstractamente efectivo.

En Copeland y Sylvan (1999), se presenta lo que los autores llaman una noción relativa de computación. El alcance de lo computable depende de los recursos disponibles. Esta es la razón de la insistencia en diferenciar entre computación que puede realizar un ser humano y computación realizada por una máquina.

Parte de esta diferencia quedaría mejor caracterizada a través de la diferencia entre efectividad y programabilidad (siendo la intersección lo computable). Pero, Copeland y Sylvan tienen en mente una versión meramente extensional de computación. Por esa razón no tienen problema en llamar computación a lo que va más allá de la tesis Ch-T. Nosotros sugerimos que se puede elucidar una caracterización intencional en la propuesta de Turing. Una manera de abordar esta caracterización es a través del estudio de las propiedades a través de la axiomatización y, por consiguiente, de la especificación de los modelos que satisfacen una noción de computación. Otra manera de abordar la cuestión es explicitando tanto la noción relacional de computación como una concepción de efectividad cuya intuición descansa en las máquinas pero que depende de ellas. Habría una noción de efectividad asociada con procedimientos – que pueden ser de carácter abstracto-.

Bibliografía

- Roland Backhouse. Algorithmic Problem Solving. Wiley Publishing, 1st edition, 2011.

- B. Jack Copeland. What is computation? Synthese, 108(3):335–59, 1996.

- B. Jack Copeland. The church-turing thesis. In Edward N. Zalta, editor, The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, summer 2015 edition, 2015.

- B. Jack Copeland and Richard Sylvan. Beyond the universal turing machine. Australasian Journal of Philosophy, 77(1):46–66, 1999.

- Nachum Dershowitz and Yuri Gurevich. A natural axiomatization of computability and proof of church's thesis. Bull. Symbolic Logic, 14(3):299–350, 09 2008.
- Jens E. Fenstad. General recursion theory : an axiomatic approach. Perspectives in Mathematical Logic. Springer Verlag, Berlin, New York, Paris, 1980.
- Nir Fresco and Marty J. Wolf. The instructional information processing account of digital computation. Synthese, 191(7):1469–1492, 2014.
- R. Gandy. Church's thesis and principles for mechanisms. In K. J. Barwise, H. J. Keisler, and K. Kunen, editors, The Kleene Symposium, volume 101, pages 123–148, 1978.
- Neil D. Jones. Computability and complexity: from a programming perspective. MIT Press, Cambridge, MA, USA, 199 7.
- J. Kennedy. Interpreting Godel: Critical Essays. Cambridge University Press, 2014.
- Gualtiero Piccinini. Computing mechanisms. Philosophy of Science, 74(4), 2007.
- Gualtiero Piccinini. Computers. Pacific Philosophical Quarterly, 89(1):32–73, 2008.
- Stewart Shapiro. Effectiveness. In Johan van Benthem, Gerhard Heinzman, M. Rebushi, and H. Visser, editors, The Age of Alternative Logics, pages 37–49. Springer, 2006.
- J. Webb. Mechanism, Mentalism and Metamathematics: An Essay on Finitism. Synthese Library. Springer Netherlands, 1980.11