¿QUÉ ES UN ALGORITMO? UNA RESPUESTA DESDE LA OBRA DE WITTGENSTEIN

WHAT IS AN ALGORITHM? A RESPONSE BASED ON WITTGENSTEIN'S WORK

Sergio Mota*
Universidad Autónoma de Madrid

Resumen: En este trabajo trato de analizar el debate en torno a la pregunta "¿qué es un algoritmo?". En la actualidad hay dos grandes enfoques representados por Y. Gurevich, quien considera que un algoritmo es una *máquina de estados abstracta*, y por Y. Moschovakis, quien considera que un algoritmo es un *recursor*. Mi propuesta es que ambas respuestas a la pregunta pueden subsumirse bajo la concepción de un algoritmo como una serie de formas, propuesta que captura la definición general de algoritmo comúnmente empleada.

Palabras clave: algoritmo, recursor, máquina de estados abstracta, Wittgenstein.

* Doctorando en el Departamento de Psicología Básica, Campus de Cantoblanco, 28049, Madrid (España), sergio.mota.v@gmail.com.

ABSTRACT: In this paper, I will try to analyze the debate on the question "What is an algorithm?" Currently, there are two main approaches represented by Y. Gurevich, who considers that an algorithm is an *abstract state machine*, and by Y. Moschovakis, who thinks that an algorithm is a *recursor*. My proposal is that both approaches to the question can be subsumed under the notion of an algorithm as a series of forms. This proposal captures the general definition of algorithm that is commonly employed.

KEYWORDS: algorithm, recursor, abstract state machine, Wittgenstein.

1. Introducción

En el *Decimocuarto Congreso Internacional de Lógica, Metodología y Filosofía de la Ciencia*, Moshe Vardi (2012) presentaba dos posibles respuestas a la pregunta "¿qué es un algoritmo?". Esas dos posibles respuestas estaban representadas por las posturas de Yuri Gurevich y Yannis Moschovakis.[1]

Por tanto, la pregunta es: *¿es un algoritmo una máquina de estados abstracta o es un recursor?* Parece necesario establecer qué definición es primaria, puesto que, matemáticamente hablando, un recursor puede modelar una máquina de estados abstracta y una máquina de estados abstracta puede modelar un recursor. Vardi propone una *dualidad algorítmica*, concluyendo que un algoritmo es tanto una máquina de estados como un recursor y que ninguna de ellas describe totalmente lo que es un algoritmo, siendo esta dualidad un principio fundamental de la ciencia de la computación. Veremos en qué medida esto es así.

En la siguiente sección presento una breve contextualización histórica, para, en la tercera sección, presentar las concepciones de Gurevich y Moschovakis.

En la cuarta sección, presento una respuesta a la cuestión basada en la obra de Wittgenstein.

[1] Vardi comienza con la siguiente pregunta: "¿No respondió ya Turing a esta cuestión?" Es decir, ¿no es *la* respuesta que un algoritmo es una máquina de Turing? Ciertamente, una máquina de Turing caracteriza la noción de algoritmo, pero no es adecuado decir que *la* respuesta, que la *única* (y *correcta*) respuesta sea que un algoritmo es una máquina de Turing. Una máquina de Turing es *una* instancia del concepto de algoritmo, no *la única* instancia.

2. Breve contextualización histórica

Es necesario exponer, siquiera brevemente, los avances llevados a cabo en la denominada teoría de la computabilidad, una rama de la lógica matemática. Como es bien sabido, la noción de algoritmo fue central durante los años 1930 en adelante, cuando la teoría de la computabilidad (antes llamada "teoría de las funciones recursivas") sufrió una expansión considerable. Diferentes autores son reconocidos como los responsables de dicha expansión, algunos considerados más relevantes o prominentes en el campo que otros. Por ejemplo, ocupan un papel preponderante Kurt Gödel, Alonzo Church o Alan Turing, mientras que otros autores también contribuyeron notablemente, como puedan ser Stephen Cole Kleene o Emil L. Post.

Así, todas las propuestas o sistemas formales establecidos a partir de los años 30 y 40 del siglo pasado son de gran relevancia para, entre otras cuestiones, el estudio de la computabilidad y para caracterizar la noción de algoritmo. Es decir, todas las propuestas llevadas a cabo por los autores antes citados tratan de caracterizar formalmente la noción de algoritmo, esto es, de procedimiento efectivo, o también procedimiento computacional o generacional, o procedimiento de decisión. De este modo, pretendían ir más allá de la noción intuitiva de algoritmo, la cual cambió cuando Hilbert, a comienzos del siglo XX, formuló el llamado 'problema de la decisión' (*Entscheidungsproblem*), que consistía en establecer un procedimiento de decisión –(*Entscheidungsverfahren*)– efectivo que permitiera determinar o decidir, en un número finito de operaciones, si una expresión o fórmula bien formada es o no válida. Obviamente, esto se relaciona con su *programa finitista*, por medio del cual pretendía probar la consistencia de la aritmética. Dicho problema fue mostrado irresoluble por varios de los autores citados anteriormente, como Gödel, Church, Turing o Post (véase Davis, 1965, 1982 para más detalles).

Así, son de crucial relevancia los avances de Church (1932) con respecto al cálculo-λ y su "primera tesis", que establecía que una función es efectivamente computable (calculable) si y sólo si es definida en el cálculo-λ (v.gr. λ-definible). Posteriormente, con la aparición de las funciones recursivas generales (Gödel, 1934), Church (1936) presentó una segunda versión de su tesis, en la que establecía que una función es efectivamente calculable si y sólo si es recursiva.[2] Es

[2] Una función *f* está definida por recursión cuando cada (nuevo) valor (de *f*) se define o especifica haciendo uso sus propios valores previamente computados para argumentos meno-

bien sabido que Gödel no estaba conforme con ninguna de las versiones de la Tesis de Church (cf. Soare, 2009). Además, según han indicado Soare (2009) o Sieg (2006), parece que la identificación hecha por Church (1936) entre funciones computables y funciones recursivas generales adolecía de errores.

Por otro lado, e independientemente de Church, Turing (1936) estableció lo que ahora se conoce como la Tesis de Turing, que indica que una función es (efectivamente) computable si y sólo si es computable por una máquina de Turing. Como señala el propio Gödel (1964), el trabajo de Turing resultó más satisfactorio que el realizado por Church, pues como el propio Gödel señala (pp. 71-72), al trabajo de Turing se debe una adecuada definición del concepto general de sistema formal. Además, como aprecia Gödel (Ibid.), también a Turing es debido un análisis del concepto de procedimiento mecánico (alias "algoritmo" o "procedimiento computacional"). Esto propició, en último término, que aceptara la tesis de Church, toda vez que se reformuló en términos de la tesis de Turing-Church.[3]

En general, se entiende que, lejos de ser totalmente nuevas estas aportaciones, son más bien la continuación de trabajos anteriores, como los de Dedekind, Frege, Russell o Hilbert, por citar algunos. Sin embargo, no se ha prestado suficiente atención, en mi opinión, a la importancia que tiene la noción formal de *forma general de serie de formas* que Wittgenstein introdujo en el *Tractatus* en 1922. Como ejemplo diré que Wittgenstein presenta un sistema formal muy similar

---

res (cf. Cutland, 1980). Siguiendo a Kleene (1943), todas las clases de funciones recursivas, esto es, las primitivas, las generales y las parciales, se construyen haciendo uso de los esquemas I-III, que aluden a la función sucesor, a las funciones contantes y a las funciones identidad como funciones iniciales, y de los esquemas IV (que es el esquema de substitución), y V (que es el esquema de definición por recursión). Bajo este esquema caen las definiciones por recursión de una variable, con parámetros y de doble recursión, aplicada a dos variables simultáneamente, como *diferentes versiones* de tal esquema. Por otro lado, el esquema VI hace referencia al esquema de *minimalización*. La clase de las funciones recursivas generales se cierra bajo I-VI, teniendo que cumplir la función definida mediante el esquema VI la exigencia de que tiene que tener al menos una solución. Al relajar esta exigencia tenemos las funciones recursivas parciales, también definidas mediante I-VI, pero en este caso, una función definida mediante VI no tiene que tener necesariamente una solución (véase Torretti, 1998, para más detalles).

[3] Desde mi punto de vista, y dado que fue Kleene y no Church quien identificó la clase de las funciones computables con la clase de las funciones recursivas parciales –que incluye a las recursivas generales y a las recursivas primitivas como totales–, sería más adecuado formular la tesis de *Turing-Kleene*, la cual se podría usar en los términos en los que se emplea la tesis de *Turing-Church*.

al de Post (1921), mediante el cual se podía determinar, dada una proposición, en un número finito de pasos, si ésta era o no una proposición de la lógica proposicional (v.gr. una tautología o verdad lógica). Sin embargo, la noción lógica de *forma general de una serie de formas* es una noción más general que puede constituir la definición formal misma de algoritmo, tal y como intentaré mostrar más abajo.

3. Gurevich y las máquinas de estados abstractas Vs. Moschovakis y los recursores

Presentado este marco general, me centraré, primero, en la aportación de Gurevich, quien parece señalar que un algoritmo es una máquina de estados abstracta –como por ejemplo una máquina de Turing– (véase Blass y Gurevich, 2007; Gurevich, 2011). Así, Gurevich (2000) está próximo a la siguiente tesis: *todo algoritmo secuencial puede ser, paso a paso, simulado por una máquina de estados abstracta apropiada*.

Un algoritmo secuencial *A* puede definirse como sigue [A_I, A_n, $_{TA}A_n$], donde A_I está por el estado inicial, A_n está por un estado arbitrario y $_{TA}A_n$ es el estado que le sigue por medio de la aplicación de map_{TA} (una operación de transformación) a A_n. Así, la serie de formas (usaré desde el principio esta expresión, aunque su aclaración y justificación se verá más abajo) A_0, A_1, A_2,…A_{n+1} puede definirse recursivamente como sigue: $A_0 = A_0$ Def; $A_{n+1} = map_{TA}(A_n)$ Def. En este sentido, las funciones que cambian de un estado de un algoritmo dado a otro estado se denominan funciones dinámicas, y un ejemplo de ello lo constituye la máquina de Turing (Gurevich, 2000, p. 9). En todo caso, un algoritmo es, siguiendo a Gurevich (op. cit., p. 15), un objeto *A* que satisface los postulados de secuenciación temporal, de estados abstractos y de exploración delimitada.[4]

[4] ¿Qué pasa con los algoritmos interactivos? ¿Cambia algo? Sí, la manera en que se ejecutan. Así, cómo se ejecuta un algoritmo no interactivo se representa mediante la serie A_1, A_2, A_3,…A_{n+1}; mientras que cómo lo hace un algoritmo interactivo se representa mediante la serie X_1, X'_1, X_2, X'_2, X_3, X'_3,…, donde el algoritmo hace los movimientos *X* y el entorno hace los movimientos *X'*. Así, la noción formal de serie de formas (ver *infra*) puede aplicarse en este caso también, de tal manera que X_0 representa el estado inicial mientras que X_1 es igual a $map_{TA}(X_0)$ ($X_1 = map_{TA}(X_0)$ Def.). Por su parte, X'_1 se obtiene a partir de X_1 por una acción del ambiente y así sucesivamente.

¿Qué hay en relación con los algoritmos no deterministas? Un algoritmo no determinista es aquel en el que dado un estado cualquiera, el algoritmo tiene varias alternativas de ejecución. Con ligeras, por no decir mínimas, modificaciones, en este caso también se mantienen

Sin embargo, como he indicado más arriba, esta no es la única identificación realizada, pues hay otro autor importante en este campo, Yannis Moschovakis, que ha hecho una identificación diferente, según la cual, un algoritmo es un recursor, esto es, una descripción recursiva construida tomando como base un conjunto de operaciones tomadas como primitivas.

Cuando Moschovakis (1998) nos habla *sobre la fundamentación de la teoría de los algoritmos,* indica que un recursor modela la estructura matemática de los algoritmos; esto es, que un algoritmo es una definición recursiva, mientras que las máquinas abstractas modelan las implementaciones (cf. Moschovkis, 2001; Moschovakis y Paschalis, 2008).

Una máquina de estados abstracta, como una máquina de Turing, es considerada por Moschovakis (1998) como un iterador (como opuesto a un recursor: mientras que éste es una definición matemática de un algoritmo, aquél modela su implementación). Desde mi punto de vista, *iteradores* y *recursores* pertenecen a niveles de análisis diferentes. Así, una máquina abstracta, como la máquina de Turing, se puede definir recursivamente, mientras que su implementación es iterativa. Comencemos, con la definición de 'iterador':

Para dos conjuntos Y y Z un iterador $iter_f$: $Y^{\sim}Z$ es del tipo (i, S, σ, T, o), donde i es el input tomado por la máquina, S es un conjunto no vacío de estados, σ es la función de transición de un estado a otro, T es el es el estado final y o es el

---

los tres postulados de secuencia temporal, de estados abstractos y de exploración delimitada, aunque este último postulado puede no ser requerido, quedando la definición como sigue: un algoritmo es un objeto que satisface los postulados de secuenciación temporal y de estados abstractos (véase Gurevich, 2000, p. 26 para más detalles).

Con respecto a los algoritmos paralelos, como opuestos a los secuenciales, Blass y Gurevich (2007) postulan la siguiente tesis: todo algoritmo paralelo se comporta de modo equivalente a una máquina de estados paralela. En este sentido, y sin entrar en mayores detalles, un algoritmo paralelo reúne los tres postulados mencionados en relación con los algoritmos secuenciales a los que se les añade otra serie de postulados (véase Blass y Gurevich, 2007, para más detalles). En todo caso, para Gurevich, un algoritmo *es* una máquina de estados abstracta. Así, cualquier algoritmo, secuencial o paralelo, se comporta de manera equivalente a una máquina de estados abstracta, lo que supone que, en último término, es necesario expresar un estado inicial, un estado de la secuencia y una operación que permita pasar de un estado a otro, esto es, una operación de transformación (Blass y Gurevich, 2007, p. 27). Por tanto, no es extraño que se pueda concebir un algoritmo paralelo trabajando de modo secuencial; esto es, pasando de una fase k a una fase k+1. Así, de acuerdo con la tesis de la máquina de estados abstracta, un algoritmo es *una máquina de Gurevich* (de estados abstracta).

output o valor que devuelve la máquina. De este modo, dado un input *x*, $x \in$ X, la secuencia de estados se puede definir recursivamente como sigue: el caso base es $S_0(x) = x$ Def., $S_{n+1}(x) = S_n(x)$, si $S_n(x) \in T$ (*i.e.*, si es el estado final) Def.; $S_{n+1} = \sigma(S_n(x))$ (*i.e.*, si la secuencia de estados continua) Def. El ser un iterador no tiene que ver con que un algoritmo pueda o no definirse recursivamente, más bien tiene que ver con la consideración de tales modelos como modelos de implementación (*i.e.*, de computación/computabilidad).

Consideremos ahora qué es un recursor. Aunque Moschovakis lo expone en varios artículos (véase 2001; Moschovakis y Paschalis, 2008), en un artículo ya clásico, (Moschovakis, 1998) lo define como sigue:

Un recursor α: X$\rightsquigarrow$W, desde un conjunto parcialmente ordenado (*partial ordered set*; i.e., *poset*) X a un conjunto W, es una tripla del tipo (D, map_{TA}, V), donde D es el dominio de α (un conjunto parcialmente ordenado inductivo), map_{TA} es la función de transición de α, y V es el valor obtenido. De este modo, Moschovakis (2001) indica que los algoritmos son definiciones recursivas o, dicho de forma similar, la teoría de los algoritmos es la teoría de las ecuaciones recursivas.[5]

Creo que, de manera más o menos explícita, es posible ver cómo tanto una máquina de estados como los recursores pueden hacer uso de definiciones recursivas. Veamos ahora qué es lo que podemos decir desde la obra de Wittgenstein.

[5] Un ejemplo de una función definida por recursión es la función adición o suma, una función recursiva primitiva: $a+0 = a$ Def., $a+(b+1) = (a+b)+1$ Def. Un ejemplo de función recursiva general es: $\varphi(0,y) = \psi(y)$ Def., $\varphi(x+1, 0) = \chi(x)$ Def., $\varphi(x+1, y+1) = \varphi(x, \varphi(x+1, y))$ Def. Como puede verse en Kleene (1952) el esquema V presenta dos versiones, aludiendo a un esquema con una variable sin parámetros (Va), o a un esquema con parámetros (Vb). Como muestra Kleene (1943), el esquema de definición por recursión (V) es también empleado para definir la clase de las funciones recursivas generales (y parciales). Dicho esto, creo que una definición por doble recursión, como la de Gödel (1934), no hace uso de un esquema distinto del esquema V, sino que más bien hace uso de otra *versión* del esquema V, aquel que permite aplicar la inducción sobre dos variables simultáneamente (p. ej. Vc). Así, se dispone de Va, Vb, Vc…

4. Una respuesta desde la obra de Wittgenstein

En este apartado expondré primero la noción de algoritmo que se puede encontrar en textos sobre los fundamentos de la informática (por ejemplo en Fernández y Sáez, 1987), o en artículos especializados (por ejemplo Shanker, 1987), para después ver la relación entre estas propuestas y el concepto formal de *forma general de una serie de formas*. Ello me permitirá formular las dos propuestas analizadas en el apartado precedente en estos términos.

Siguiendo a Fernández y Sáez (1987, pp. 311-312), hay diferentes definiciones de 'algoritmo'. Por ejemplo, Shanker (1987, p. 632) indica que un algoritmo puede entenderse como una secuencia definida de reglas (operaciones) que especifica cómo producir un resultado (output) desde un input dado en un número finito de pasos.

En consonancia con esta definición, Fernández y Sáez (1987, p. 313) presentan la siguiente cuádrupla, que expresa la definición de algoritmo desde la teoría de conjuntos: $A = <Q, E, S, F>$, donde Q es el conjunto de todos los elementos simples y K-fórmulas que pueden describir el cálculo, E es un subconjunto de Q que hace referencia a los datos de entrada, S también es un subconjunto de Q cuyos elementos son los resultados y F se refiere a la regla del cálculo. Esta regla, a partir de un elemento q_0, genera una sucesión $q_1, q_2, q_3, \ldots$ tal que: $q_{n+1} = F(q_n)$, donde $n \in N$, $q_0 \in E \subset Q$.

Creo que se aprecia con claridad la relación entre esta exposición y la presentada en el apartado precedente, con respecto, por ejemplo, a la secuencia de estados de una máquina abstracta (como la de Turing). Pues bien, en 1922, Wittgenstein, en el *Tractatus Logico-Philosophicus*, definió el concepto formal de *forma general de una serie de formas*.[6] Un concepto formal es, o está expresado por, una variable (4.127): "Toda variable es el signo de un concepto formal" (4.1271). La explicación del uso de esa variable define el concepto formal. De este modo "[u]n concepto formal está ya dado en cuanto se da un objeto que cae bajo él" (4.12721). En 4.1273, Wittgenstein indica que "[e]l término general de una serie de formas sólo puede expresarse mediante una variable" (cf. 4.1272, 4.1252, 4.126).

[6] Una serie de formas es una serie ordenada por relaciones (o propiedades) internas (en oposición a externas o contingentes) –véase Wittgenstein (1922, 4.1252).

¿Cuál es el término general de una serie de formas? En 5.2522, Wittgenstein escribe que el término general de una serie de formas *a*, *O'a*, *O'O'a*,… es [*a*, *x*, *O'x*], donde *a* es el comienzo de la serie, *x* es un término de la serie generado, y *O'x* es el resultado de aplicar *O'* a un valor previamente computado *x*.

Queda, pues, analizar cómo la forma general de una serie de formas muestra y recoge lo esencial de las formulaciones expresadas en términos de iteradores, recursores y a la definición de algoritmo dada por Fernández y Sáez.

Dicho esto, un algoritmo es una serie de formas expresada por medio de la siguiente tripla: $[\tilde{\sigma}, \tilde{\varepsilon}, \acute{O}(\tilde{\varepsilon})]$, donde $\tilde{\sigma}$ expresa un input o un conjunto de ítems iniciales, un dominio en el que no es necesario que todos los ítems que se toman como argumentos estén definidos; esto es, que les corresponda un valor. $\tilde{\varepsilon}$ está por un subconjunto de esos ítems iniciales, un subconjunto del dominio o de los datos de entrada. Finalmente, $\acute{O}(\tilde{\varepsilon})$ indica tanto un valor o resultado, como la operación de transición que permite pasar de un valor a otro, expresada por medio de $\acute{O}(\ )$. Es claro que una secuencia como $\tilde{\varepsilon}$, $\acute{O}^{(n)}(\tilde{\varepsilon})$, $\acute{O}^{(n+1)}(\tilde{\varepsilon})$… captura una secuencia de estados como $S_0(\tilde{\varepsilon})$, $S_n(\tilde{\varepsilon})$, $S_{n+1}(\tilde{\varepsilon})$,… Así, el paso de $\tilde{\varepsilon}$ $(=\acute{O}^{(0)}(\tilde{\varepsilon}))$ a $\acute{O}(\tilde{\varepsilon})$ muestra la transición de un estado a otro y, por tanto, que el conjunto de estados no está vacío. Además, ambos pueden instanciar el estado final. De este modo, un 'estado', que no tiene por qué tener ningún sentido psicológico, es una instancia de 'forma', una noción lógico-matemática.

La tripla recién expuesta recoge lo esencial, lo común, a las diferentes propuestas para definir un algoritmo. Evidentemente, puede ser expresada por medio de una definición recursiva (cf. Wittgenstein, 1974, 36):[7]

$\acute{O}^{(0)}\tilde{\varepsilon} = \tilde{\varepsilon}$ Def.;

$\acute{O}^{(n)\prime}\tilde{\varepsilon} = \acute{O}'\acute{O}^{(n-1)}\tilde{\varepsilon}$ Def.

Así, este análisis muestra, en mi opinión, que no hay tal dualismo algorítmico. Es decir, estoy de acuerdo con Vardi en que ninguna de las dos respuestas, en términos de máquinas abstractas o en términos de recursores, describe totalmente lo que es un algoritmo, pero no comparto que esta dualidad sea un principio

[7] Un ejemplo: la forma general de una serie de formas '[0, ξ, ξ + 1]', expresa una regla gramatical del tipo '*a*+0 = *a*; *a*+ (ξ + 1) = (*a*+ξ)+1'. Éste es un ejemplo claro de recursor.

fundamental de la ciencia de la computación. Como puede apreciarse, tanto una máquina como un recursor no son sino instancias de una serie de formas, estando sus términos (o estados) ordenados por relaciones –o propiedades– internas o formales. Esto es, cada (nuevo) término resulta de aplicar una operación de transición a un término previamente computado.

Por último, quisiera resaltar que un algoritmo, desde la perspectiva wittgensteiniana que defiendo aquí, no debería considerarse un *descubrimiento* matemático de una realidad empírica o platónica, sino, más bien, una *construcción* o *invención* matemática.

Aunque no pretendo establecer que este análisis es el único posible o el único correcto, sí quiero subrayar la relevancia de Wittgenstein para el análisis del concepto formal de algoritmo. Esto, creo, no ha sido suficientemente reconocido, al menos en los textos citados, algunos de los cuales se emplean en el análisis actual del concepto. Creo que la aportación de Wittgenstein es, en todo caso, interesante y relevante.

Mi gratitud a los revisores anónimos por su lectura y útiles comentarios.

Bibliografía

BLASS, Andreas. & GUREVICH, Yuri. (2007). "Abstract state machines capture parallel algorithms: correction and extension". *ACM Transactions on Computational Logic*, *V*, pp. 1-29.

CHURCH, Alonzo. (1932). "A set of postulates for the foundation of logic". *The Annals of Mathematics*, Vol. 33, pp. 346-366.

— (1936). "An unsolvable problem of elementary number theory". *The undecidable*. Ed. Davis, M. New York: Raven Press, pp. 88-107.

CUTLAND, Nigel. (1980). *Computability: an introduction to recursive function theory*. Cambridge: Cambridge University Press.

DAVIS, Martin. (1965). *The undecidable*. New York: Raven Press.

— (1982). "Why Gödel didn't have Church Thesis". *Information and Control*, Vol. 54, pp. 3-24.

FERNÁNDEZ, Gregorio. y SÁEZ, Fernando. (1987). *Fundamentos de informática*. Madrid: Alianza

GÖDEL, Kurt. (1934). "Sobre sentencias indecidibles de los sistemas formales matemáticos". *Obras Completas*. Ed. Mosterín, J. Madrid: Alianza, pp. 167-196.

— (1964). "Postscriptum to Gödel 1931". *The undecidable*. Ed. Davis, M. New York: Raven Press, pp. 71-73.

GUREVICH, Yuri. (2000). "Sequential abstract state machines capture sequential algorithms". *ACM Transactions on Computational Logic*, Vol. 1, pp. 77-111.

— (2011). "What is an algorithm?". *Technical report MSR-TR-2011-116*.

KLEENE, Stephen Cole. (1943). "Recursive predicates and quantifiers". *Transactions of the American Mathematical Society*, Vol. 53, pp. 41-73.

— (1952). *Introduction to metamathematics*. Amsterdam: North-Holland Publishing.

MOSCHOVAKIS, Yiannis. & PASCHALIS, Vasilis. (2008). "Elementary algorithms and their implementations". *New computational paradigms*. Eds. Cooper, S.B., Löwe, B. & Sorbi, A. Springer, pp. 87-118.

MOSCHOVAKIS, Yiannis. (1998). "On founding the theory of algorithms". *Truth in mathematics*. Eds. Dales, H.G. & Oliveri G. Oxford University Press, pp. 71-104.

— (2001). "What is an algorithm?". *Mathematics Unlimited: 2001 and Beyond*. Eds. Engquist B. & Schmid, W. Berlin: Springer, pp. 919-936.

POST, Emil. (1921). "Introduction to a general theory of elementary propositions". *American Journal of Mathematics*, Vol. 43, pp. 163-185.

SHANKER, Stuart. (1987). "Wittgenstein versus Turing on nature of Church's thesis". *Notre Dame Journal of Formal Logic*, Vol. 28, pp. 615-649.

SIEG, Wilfried. (2006). "Gödel on computability". *Philosphia Mathematica*, III, pp. 189-207.

SOARE, Robert. (2009). "Turing oracles machines, online computing, and three displacements in computability theory". *Annals of Pure and Applied Logic*, Vol. 160, pp. 368-399.

TORRETTI, Roberto. (1998). *El paraíso de Cantor. La tradición conjuntista en la filosofía matemática*. Santiago de Chile: Editorial Universitaria.

TURING, Alan. (1936). "On computable numbers, with an application to the Entscheidungsproblem". *The undecidable*. Ed. Davis, M. New York: Raven Press, pp. 116-151.

VARDI, Moshe (2012). "What is an algorithm?". *Communications of the ACM*, Vol. 5(3), p. 5.

WITTGENSTEIN, Ludwig. (1922). *Tractatus logico-philosophicus*. Londres: Routledge.

— (1974). *Philosophical grammar*. Oxford: Blackwell.

Enviado: 30/06/2015

Aceptado: 24/08/2015