Lenguajes y arquitecturas

Pío García*
Javier Blanco**

Introducción

Hemos propuesto en trabajos anteriores a la efectividad y programabilidad como dimensiones adecuadas para abordar la noción de computación. En este trabajo queremos avanzar en la sugerencia de que dichas dimensiones pueden representar una perspectiva ventajosa para comprender aspectos conceptuales e históricos de la computación.

Como es bien sabido, en la búsqueda de lograr una caracterización no ambigua ni circular de procedimiento efectivo, Alan Turing propuso la noción que hoy consideramos estándar de computación. La descripción de los elementos integrantes de una *máquina de Turing* vendría a dar cuenta del comportamiento de un ser humano actuando mecánicamente. De esta manera, el aspecto mecánico aparecía como constitutivo de lo computacional.

Ahora bien, la discusión sobre lo que constituía un procedimiento efectivo estaba, a principios del siglo XX, circunscripta a la lógica y los fundamentos de la matemática. Pero la tradición de las máquinas que calculan, como la pascalina o los motores de Babbage, y de las máquinas que decodifican, como el telar de Jacquard, se originó y desarrolló de manera bastante independiente. Aunque luego, en el siglo XX, terminarían por confluir en la construcción de las primeras computadoras.

La efectividad y programabilidad permiten, desde nuestra perspectiva, describir las propiedades fundamentales de la tradición de las máquinas de cálculo y aquellas que tenían algún sistema de codificación. La efectividad de estas máquinas estaba asociada con diversas propiedades y características: desde la confiabilidad de los materiales hasta la organización de los sub-sistemas para ganar en automatización de procesos. También, por

* Universidad Nacional de Córdoba (UNC), Facultad de Filosofía y Humanidades (FFyH), Córdoba, Argentina.

** Universidad Nacional de Córdoba (UNC), Facultad de Matemática, Física, Astronomía y Computación (FAMAF), Córdoba, Argentina
piogarcia@ffyh.unc.edu.ar

supuesto, estaba asociada con una de las promesas por las cuales se construían máquinas: una mayor velocidad en las tareas mecanizadas. Efectividad aquí, entonces, está vinculada con aquellos aspectos que hacen que una máquina sea veloz, autónoma y confiable.

A su vez, un aspecto central de estas máquinas era el grado en el cual se podían codificar comportamientos o las tareas para las cuales eran construidas. El que un sistema sea programable y en qué medida lo era, constituía una propiedad definitoria no sólo para proto-computadoras —en el sentido temporal— como las de Babbage sino para dispositivos automatizados como el telar de Jacquard. Una de las innovaciones características en este tipo de telar era la codificación de patrones en un sistema de tarjetas perforadas.

Dentro de este esquema era esperable, y hasta natural, asociar la efectividad de las máquinas al *hardware* (a aquellos aspectos más directamente vinculados con lo físico) y la programabilidad con el *software* o con los procedimientos de codificación.

En gran medida, la propuesta de Turing vino a «atar» de una manera original la efectividad y la programabilidad a través de las ideas de máquina de Turing y máquina universal. Pero esta propuesta estuvo en un principio circunscripta al contexto lógico y matemático del cual surgió.

Nuestra propuesta es que la discriminación entre estos aspectos no solo puede servir para iluminar aspectos *conceptuales* de la noción de computación sino para entender algunos aspectos de su *historia*. En este trabajo vamos a tomar como ejemplo algunos hitos de la constitución de la noción de arquitectura y de la actividad de programar.

Pero antes de abordar esta cuestión creemos que puede ser útil especificar en qué sentido las propuestas originales de Turing podrían complementarse señalando los alcances de la noción de computación, lo que habitualmente se ha tratado bajo la denominación de la tesis Church-Turing.

Gandy y las tesis Church-Turing

La cuestión de qué está involucrado en la noción de computación podría ser investigada, asimismo, a partir de distinguir entre perspectivas diferentes que a veces, histórica y conceptualmente, han sido vistas como equivalentes. En un texto de 1980 —*Church's thesis and principles of mechanism*— Gandy distingue entre tres versiones de la llamada tesis

Church-Turing. El primer enunciado es la tesis de Church: "lo que es efectivamente calculable es computable" (Gandy, 1980, p. 123). El segundo enunciado es llamado el teorema T (por Turing): "lo que puede ser calculado por un ser humano abstracto trabajando de una manera rutinaria es computable" (Gandy, 1980, p. 124). Finalmente, el tercer enunciado es el que Gandy llama Tesis M: "lo que puede ser calculado por una máquina es computable". Lo calculable, lo realizable por un ser humano bajo ciertas restricciones y aquello que puede hacer una máquina son contextos todos ellos relevantes para entender a la computación, pero no equivalentes.

Estas versiones de la tesis Church-Turing suelen plantearse tomando en consideración la idea de procedimiento efectivo, la cual se entiende en términos de una máquina de Turing. A partir de lo reseñado por Gandy y de los recientes comentarios de Copeland y Shagrir (2007, 2019), se podrían hacer las siguientes consideraciones. En primer lugar, en la ciencia computacional contemporánea los algoritmos —o procedimientos efectivos— están asociados con *máquinas* y no con seres humanos (Copeland & Shagrir, 2019, p. 68). Entonces, ¿qué diferencias habría entre el tipo de restricción que encontramos en los seres humanos y en las máquinas?:

> el argumento [de] Turing, contiene "pasos cruciales . . . donde él [Turing] apela al hecho de que el cálculo lo está realizando un ser humano". Por ejemplo, Turing supuso que "un ser humano solo puede escribir un símbolo a la vez", y Gandy señaló que esta suposición no se puede transferir a una máquina paralela que "imprime un número arbitrario de símbolos simultáneamente". (Copeland & Shagrir, 2019, p. 69)

En segundo lugar, como es evidente, los algoritmos computacionales actuales están muy lejos de poder ser descriptos de manera *directa* por una máquina de Turing.

> La máquina de Turing [no] puede duplicar (a diferencia de simular) *El juego de la vida* de John Conway, donde, a diferencia de una máquina de Turing, *cada celda se actualiza simultáneamente* [cursivas nuestras]. (Copeland & Shagrir, 2019, p. 68)

Siguiendo este razonamiento, Gandy, en su análisis de la idea mecánica de computación incluye el *paralelismo* como una propiedad de computadoras que no toman a los seres humanos como modelo y límite. En este sentido, más que una descripción completa de lo que es un algoritmo, una máquina de Turing representa un paso, crucial, pero en cierto sentido in-

suficiente, para la construcción de una noción adecuada de computación. Evidentemente el paso conceptual importante involucra comprender de qué manera un programa se relaciona con sus datos en diferentes niveles de análisis; lo cual es, en cierto sentido, equivalente a la idea de máquina universal. Sin embargo, a pesar de su importancia conceptual, y como vimos en el ejemplo presentado por Gandy, tampoco parece suficiente el mero señalamiento este este paso: hace falta la *especificación* de la manera en la cual esta idea puede implementarse.

Creemos que la discusión sobre la manera de articular las arquitecturas computacionales y el desarrollo de la actividad de programación en los '40 y '50 del siglo pasado pueden entenderse como parte de esta tarea de especificación.

En concreto, nuestra estrategia para las secciones siguientes será mostrar cómo en un principio los aspectos ligados a la efectividad en la tradición de las máquinas, a saber, velocidad, automatización y confiabilidad, se trasladan casi de manera directa a la construcción y al uso de las primeras computadoras. De esta manera, se entendía que una máquina solo puede ser más efectiva si hay una mejora cuantitativa en el *hardware.* Las primeras computadoras estaban en la senda marcada por la tradición de las máquinas, en el doble aspecto señalado más arriba.

Pero una progresiva comprensión de los problemas involucrados en la computación –a través del trabajo de von Neumann– permitió el surgimiento de un tipo de respuesta, una nueva arquitectura, que se traducía, por supuesto, en un nuevo *hardware,* pero cuya motivación era más bien conceptual. A su vez, problemas típicamente asociados con la efectividad como la mayor velocidad y la automatización comenzaron a ser abordados a partir de la codificación. De esta manera, lo que en la tradición de las máquinas era un aspecto principalmente asociado con la plasticidad o la flexibilidad, pasó a ser una estrategia o un camino para lograr mayor efectividad. Y es justamente esta relación original entre efectividad, de la manera en la cual la hemos propuesto, y programabilidad, que desdibuja el límite entre *software* y *hardware*, lo que, a nuestro juicio, caracteriza a la computación.

Veamos a continuación cómo puede leerse a partir de esta clave de análisis algunos aspectos históricos de la computación.

Arquitecturas

La historia de las arquitecturas computacionales difícilmente pueda ser contada de manera lineal y clara. Las máquinas del siglo XIX, como el motor diferencial y el analítico de Babbage, realizaron aportes relevantes en cuanto a la organización, sin embargo, su importancia fue reconocida de manera tardía.[1]

Hay otros ejemplos del siglo XX como las máquinas construidas por Konrad Zuse desde 1935, *Z1*, *Z2* y *Z3*, que pueden ser consideradas como las primeras computadoras electrónicas controladas por programas. Pero luego de ser destruidas en 1944, su relevancia histórica fue reconocida recién muchos años después. Por tanto, en este trabajo vamos a concentrarnos en las arquitecturas computacionales Harvard y von Neumann.

En un trabajo anterior hemos presentado los orígenes de la computación científica a través del trabajo de Howard Aiken y Leslie Comrie (Blanco & García, 2020). Aquí solo hace falta recordar la máquina que representa la arquitectura Harvard: la Mark I. A finales de los 1930, Aiken visita varias empresas con la idea de construir una máquina de cálculo.

Finalmente, IBM se interesa por el proyecto y entre 1939 y 1943 se construye la ASCC, Calculadora Secuencial de Control Automático [*Automatic Sequence Controlled Calculator*], llamada luego, cuando se instala en Harvard, la Mark I.

Esta máquina fue utilizada para cálculos "balísticos" (Priestley, 2011, p. 102). A esta computadora le siguieron una serie de máquinas diseñadas y construidas en Harvard en los '40 y '50 del siglo pasado (las Mark II, III y IV, de las cuales la Mark IV era completamente electrónica).

La complejidad técnica de esta máquina era importante. Tenía más de 700.000 componentes electromagnéticos y pesaba más de cuatro toneladas. Lo importante, para nosotros, es que en esta máquina había una separación clara entre *datos* e *instrucciones*. No solo había una separación física, sino que los *formatos* en los cuales estaban codificados los datos e instrucciones eran diferentes.

Una cinta separada podría contener números para entrada, pero los formatos de cinta no eran intercambiables. Las instrucciones no se podían ejecutar desde los registros de almacenamiento (Priestley, 2011).

[1] Por esta razón presentamos la efectividad y la programabilidad en la tradición de las máquinas de esta manera. Con el aporte de Turing podría leerse en retrospectiva de una manera diferente el trabajo de Babbage.

Esta estructura particular que caracteriza a una arquitectura Harvard estaba motivada por una razón de peso ya que dicha estructura permitía acceder de manera *simultánea* a los datos y al programa. La mayor *velocidad* era la razón que motivaba la apuesta por esta organización.

La otra arquitectura importante de esta época surgió en la Escuela Moore de Ingeniería Eléctrica de la Universidad de Pensilvania. A mediados de los años cuarenta del siglo pasado, J. Presper Eckert y John Mauchly construyeron una máquina automática de cálculo electrónica que llamaron ENIAC. Una de las funciones principales de esta máquina era calcular las tablas balísticas para el Laboratorio de Investigación correspondiente de la Armada (Aspray, 1990). Eckert y Mauchly al hacerse conscientes de las limitaciones de la ENIAC comienzan a discutir una versión mejorada de esta computadora: la EDVAC.

En 1944 había comenzado a participar de algunas reuniones en la Escuela Moore, John von Neumann, quien quería usar recursos de cálculo para su proyecto de la bomba atómica. De manera inmediata se compromete con la discusión del diseño de la EDVAC. Así, von Neumann escribió un reporte en 1945 –*First Draft of a Report on the EDVAC.* Este reporte suele ser considerado como la primera publicación del concepto de "programa almacenado".

El aspecto definitorio de la arquitectura von Neumann es que la distinción entre *datos* y *programa* es relativizada desde el *hardware* mismo. Esta relativización implica un paso atrás, al menos en un sentido inmediato, en la búsqueda de mayor velocidad, mayor efectividad podríamos decir, para obtener una flexibilidad cualitativamente significativa. Evidentemente, la noción de máquina universal es la que permite describir este paso.[2]

Bajo esta perspectiva aparece una ventaja cualitativa de la arquitectura von Neumann con respecto a la de Harvard. Pero hay un aspecto del trabajo concreto de los investigadores en la Mark I, la arquitectura Harvard, que queremos destacar: la creciente importancia de la programación para solucionar problemas que en principio parecían ser de otra clase o abordables con otras herramientas. Además, como veremos, muchos de los avances tecnológicos relacionados con la programación no apuntaban solo a la *plasticidad* o flexibilidad del sistema, sino a lo que hemos llamado antes la *efectividad*, representada en la mayoría de los casos por la búsqueda

[2] Michael Godfrey cita en su introducción al comentario del *First Draft* que Turing en su reporte sobre la ACE cita el informe de von Neumann como un ejemplo de máquina de propósito general.

de mayor velocidad. También aparece aquí como objetivo la búsqueda de una mayor automatización. Todos estos aspectos muestran una creciente influencia de una manera de entender la efectividad y la programabilidad en términos computacionales, esto es en términos de *integración*.

Para mostrar estos aspectos y simplificar la presentación vamos a concentrarnos en el trabajo pionero de Grace Hopper en programación.

Programación

Cuando Hopper fue enviada por la Armada a trabajar con Aiken en la Mark I, se dio cuenta de que para poder programar esa máquina necesitaba entender la *organización física* de la computadora (Beyer, 2009, p. 47). Hopper tuvo que aprender electrónica y estudiar los planos de circuitos; lo cual no representaba una tarea sencilla dado, como vimos, el tamaño y complejidad del aparato. Es más, si consultamos el manual de operaciones que escribió Hopper, veremos que en aquellos lugares en donde se explica cómo programar la computadora aparecen, habitualmente, diagramas de circuitos.

Hay que señalar que estos límites de la Mark I, en cuanto a su configuración para cómputos específicos, no dependían de su carácter *mecánico*. Algo similar ocurría con los primeros sistemas computacionales *electrónicos*. La computadora ENIAC, de la Universidad de Pensilvania era mucho más rápida por tener un sistema de tubos de vacío. Pero su «lógica» era la misma: codificar la máquina era equivalente a la manipulación del *hardware* (Beyer, 2009, p. 51). A fines de los cuarenta Eckert y Mauchly contratan a seis mujeres del Laboratorio de Investigación Balística de la Armada para trabajar en la ENIAC.[3] Al igual que Hopper, estas mujeres pronto se dieron cuenta que para programar la ENIAC tenían que conocer el *hardware*.[4]

Pero Hopper había desarrollado en poco tiempo, una estrategia de trabajo con la computadora mecánica *Mark I* que privilegiaba la codificación. Lo cual contrastaba con la aproximación que se llevaba adelante con otras computadoras más avanzadas desde el punto de vista del *hardware*.

---

[3] Los otros programadores de la Eniac fueron Kay McNulty, Betty Jennings, Betty Snyder, Marlyn Wescoff, Fran Bilas y Ruth Lichterman.

[4] Este aspecto se reflejaba en una división de trabajo clara: los oficiales eran los *codificadores* y los reclutas eran los *operadores* que cableaban la máquina.

En palabras de la propia Hopper:

> Cuando visité [la gente que trabajaba en] ENIAC, el tremendo contraste entre Harvard y las actividades de Eckert y Mauchly fue la *programación*. Lo de ENIAC era: conectar piezas y esencialmente construir una computadora *especial* para cada trabajo, y nosotros estábamos acostumbrados al concepto de programación y de controlarlo a través de nuestro programa. (citada en Beyer, 2009, p. 52)

De esta manera, aquí estaba en consideración también, la apuesta por máquinas de propósito general y de propósito especial que caracterizó los inicios de la computación científica.

Ahora bien, ¿cómo había llegado Hopper a esta idea de la relevancia de la «programación»? Ella comenzó a trabajar en la Mark I durante los últimos años de la segunda guerra mundial y los requerimientos del ejército eran apremiantes. Sin embargo, solo Hopper y Bob Campbell eran los encargados de manejar la máquina. Y manejar la máquina significaba, como ya vimos, cambiar la organización de los cables que constituían la computadora. Dada la urgencia y cantidad de demanda comenzaron a construir «rutinas» de código con una aproximación sistemática para ahorrar en trabajo de modificación del cableado físico (Beyer, 2009, p. 53).

Ahora bien, la *Mark I* era demasiado lenta para la demanda durante la guerra. Funcionaba 24/7 pero no era suficiente. Por esta razón, Hopper y Bloch comenzaron en 1944 a optimizar las rutinas del *software* para minimizar la *cantidad de ciclos* necesarios para realizar cada computo. Así, se redujo en un 36 por ciento el tiempo de ejecución de los programas. La búsqueda era de programas más *eficientes*. La contrapartida fue que los programas se hicieron cada vez más complejos y difíciles de entender.

Otra cuestión era el formato del resultado. En un principio la Mark I imprimía un conjunto de números que tenía que ser interpretado por un operario quien, utilizando una máquina de escribir «traducía» el resultado de la computadora. A Hopper se le ocurrió *automatizar* ese proceso con un programa para traducir el resultado.

Así, como muestran estos ejemplos, a los problemas de mayor velocidad y de automatización se los comenzó a atacar con soluciones de codificación. Evidentemente, la importancia de la aproximación de la automatización a través de la codificación podía verse también en el trabajo con otras computadoras de la época. En 1950 Mauchly comenzó a utilizar en la UNIVAC lo que llamó «códigos breves» [*Short codes*], los cuales consistían

en ordenes básicas o "código especial elegido para simplificar el trabajo del programador humano y dejar de lado mucho del trabajo tedioso de codificación detallada en la computadora" (Priestley, 2011, p. 187).

Pero esta manera de trabajar tenía como contrapartida, tal como señalamos más arriba, una complejización del código. La *accesibilidad epistémica*, un problema hasta ahora típicamente asociado con la complejidad de la máquina física,[5] aparece en el contexto de la tarea de codificación. Además, la multiplicidad de códigos asociados con máquinas específicas venía a agravar este problema.

El señalamiento de estos problemas provocó diferentes respuestas. Por ejemplo, a fines de los cincuenta se propone un lenguaje de programación, Algol 58, con la finalidad de colaborar en la estandarización. Uno de los objetivos de este lenguaje era la "expresión y comunicación de los algoritmos" (Priestley, 2011, p. 206). Además, el lenguaje debía ser «trasladable» mecánicamente en programas de máquina. Sin embargo, un aspecto inusual, para la época, era que este lenguaje no hacía referencia a una máquina particular. Las dificultades para implementar Algol 58 llevaron al desarrollo de Algol 60 cuyo objetivo principal, ahora mucho más explícito, era la *comunicación de los algoritmos*.

Algunos hitos del trabajo posterior de Hopper muestran otro aspecto interesante. Cuando Hopper comenzó a escribir subrutinas para la UNIVAC, a partir de 1951, de manera semejante a las implementadas en la Mark I, su trabajo derivó en la construcción del sistema A-0. Desde una perspectiva contemporánea el sistema A-0 es considerado como el primer compilador. Pero, en esa época, no era sencillo conceptualizar qué se estaba haciendo. Priestley (2011) señala que había varias interpretaciones sobre este tipo de programas. En primer lugar, se interpretaban como *extensiones* del código de máquina. En segundo lugar, una interpretación habitual era como una *traducción de un tipo de lenguaje* al lenguaje de máquina. Otra interpretación era verlo como una *traducción o extensión de la máquina misma*.

Esta última idea es la que también había propuesto Turing para entender la clase de algoritmo involucrado aquí. Turing lo llamaba «rutina interpretativa» y lo describía como aquellos programas que "permitían a la computadora convertirse en una máquina que usa una instrucción

[5] La tapa de la revista especializada en computación *Communications of the ACM* de 1961 llevaba la imagen de la torre de Babel con los nombres de los lenguajes de programación más populares de la época.

de código diferente a la originariamente requerida para el diseño de la computadora" (Priestley, 2011, p. 190). Esta referencia sirve para poner de manifiesto cómo el problema de la construcción de un tipo de programa que permita comunicarse de manera más simple con el lenguaje de máquina, desde un nivel conceptual, es un tipo de problema similar al que abordaban aquellos que proponían las arquitecturas von Neumann. Y es finalmente el mismo tipo de solución que propuso Turing, aunque desde una perspectiva más general, con su máquina universal.

Consideraciones finales

En este trabajo hemos intentado mostrar las ventajas que tendría analizar aspectos conceptuales e históricos de la computación a partir de las dimensiones de la efectividad y la programabilidad. Estas dimensiones aparecen configuradas de manera diferente en lo que hemos llamado la tradición de las máquinas y en la tradición de las computadoras.

Sin embargo, las primeras computadoras construidas en la primera mitad del siglo XX muestran una continuidad importante con la tradición de las máquinas. Esto se evidencia tanto en las arquitecturas, ejemplificada en la Mark I de Harvard, como en la práctica de la programación entendida en términos de modificación de cableado físico de las máquinas. Además, esta continuidad permite ver el paulatino desplazamiento de problemas típicamente asociados con la efectividad de las máquinas al terreno de la programación. Este desplazamiento no fue homogéneo ni directo en el caso de las arquitecturas y los lenguajes –como vimos con el caso de la importancia cada vez mayor de la programación en la Mark I. Este aspecto histórico permite destacar con mayor fuerza el creciente desdibujamiento entre los límites de los problemas tradicionalmente asociados con el *hardware* y el *software*. Lo que venía a representar la solución teórica de Turing en 1936, en lo que hemos llamado en otros trabajos una noción relacional de computación, se fue configurando como un horizonte, no siempre claro, de solución de problemas. En este camino de solución de problemas concretos, de velocidad, automatización y confiabilidad se fueron reconfigurando las notas características de lo que significaba efectividad y programabilidad en la tradición de las máquinas.

Referencias

Aspray, W. (1990). The stored program concept. *IEEE Spectrum*, *27*(9), 51.

Beyer, K. (2009). *Grace Hopper and the invention of the information age*. Cambridge, MA.: MIT Press.

Blanco, J., & García, P. (2020). En torno a las nociones de efectividad y programabilidad para comprender la noción de computación. En M. O'Lery, L. Federico, & Y. Ariza (Eds.), *Filosofía e historia de la ciencia en el cono sur. Selección de trabajos del XI Encuentro de la Asociación de Filosofía e Historia de la Ciencia del Cono Sur* (pp. 102-109). São Carlos-Buenos Aires: AFHIC.

Copeland, B. J., & Shagrir, O. (2007). Physical computation: How general are Gandy's principles for mechanisms? *Minds and Machines*, *17*(2), 217-231.

Copeland B. J., & Shagrir, O. (2019). The Church-Turing thesis: logical limit or breachable barrier? *Communications of the ACM*, 62, 1, 66-74

Gandy, R. (1980). Church's thesis and principles for mechanisms. En J. Barwise, H. J. Keisler, & K. Kunen (Eds.), *Studies in logic and the foundations of mathematics* (pp. 123-148). Elsevier.

Priestley, M. (2011). *A science of operations: machines, logic and the invention of programming*. Springer Science & Business Media.