MÁQUINAS DE TURING DINÁMICAS

Informe final de investigación

ANDRÉS SICARD RAMÍREZ

DEPARTAMENTO DE CIENCIAS BÁSICAS
ESCUELA DE CIENCIAS Y HUMANIDADES
UNIVERSIDAD EAFIT
MEDELLÍN
1998

AGRADECIMIENTOS

A la universidad EAFIT por la beca otorgada para realizar esta maestría y por la aprobación del proyecto de investigación: “Máquinas de Turing dinámicas”, que permitió obtener los recursos necesarios para realizar esta tesis. A nuestro asesor Félix Londoño por sus recomendaciones metodológicas, motivaciones constantes e interés demostrado durante la gestación y desarrollo de esta tesis. A Andrea Cano por su compañía y comprensión. A Antonio José Ramírez por habernos enviado los artículos originales de Alan Turing. Y en general a todas aquellas personas que nos estimularon y nos soportaron durante la realización de este proyecto, pero también a aquellas personas que nos desestimularon, porque aunque en algunos aspectos estaban acertados, su dogmatismo nos motivo a seguir adelante.

LISTA DE FIGURAS

TABLA DE CONTENIDOS

INTRODUCCIÓN

Este documento presenta la historia y el desarrollo de una idea: La máquina de Turing dinámica (MTD). Existen varias alternativas de narrar una historia; en esta ocasión la narración respetará el orden cronológico de los acontecimientos y resultados obtenidos, aunque a la luz de estos resultados, la historia podría ser descrita de manera muy diferente. El autor desea conducir al lector por el mismo camino que transitó, durante el desarrollo de su idea.

El capítulo 1 construye el marco teórico necesario para que la noción de MTD adquiera un contexto científico–filosófico–teórico. Preguntas como: ¿Cuáles son las propiedades de la computabilidad?, ¿Es posible ampliar la definición de computabilidad?, ¿En caso de ser posible esta ampliación, qué características debería tener?, ¿Cuál es la relación de esta posible ampliación con la tesis de Church–Turing?, etc., son discutidas (implícita o explícitamente). Además se presentan (algunos) los argumentos de Church en relación con lo que significaría ampliar la noción de computabilidad y el porque él considera que es muy poco probable que esto pueda realizarse.

El capítulo 2 presenta la gestación de la idea de la MTD. Esta presentación se realiza debido al convencimiento de la importancia de las ideas como del desarrollo de las mismas. La historia de la ciencia con frecuencia presenta las ideas científicas como salidas de la nada o a partir de hechos tan simples como la caída de una manzana; creadas por seres excepcionales, seres atemporales, aculturales, ahistoricos. Se tendría una visión muy diferente de la ciencia, si la historia no maquillara la gestación de las ideas y en particular no ocultara los fracasos, los errores y el matiz personal que rodean a las mismas.

El capítulo 3 describe que es una MTD. A partir de la descripción realizada en el capítulo 3, el capítulo 4 se pregunta por las consecuencias de este constructo teórico. En particular, se pregunta por la relación que existe entre una MT y una MTD; relación analizada desde un único punto de vista: la relación de potencia entre las máquinas.

El capítulo 5 es un capítulo formal. Se presentan los elementos necesarios para construir una MTD. Para algunos casos se construyen estos elementos a manera de prueba de la viabilidad de la construcción de la MTD. Finalmente el capítulo 6 presenta los resultados obtenidos. Resultados que no son evidentes ni esperados a partir de la historia narrada, pero al fin de cuentas, esta es la historia de la MTD.

Se espera que el lector tenga buenos conocimientos en teoría de la computabilidad (clásica), particularmente en aspectos tales como: máquina de Turing, máquina universal de Turing, m–funciones y codificación para las máquinas de Turing.

1 MARCO TEÓRICO

Robert Soare estable el espacio donde opera el concepto de computabilidad, con la siguiente descripción:

"A *computation* is a process whereby we proceed from initially given **objects**, called *inputs*, according to a fixed set of rules, called a *program* , *procedure* , or *algorithm* , through a series of *steps* and arrive at the end of these steps with a final result, called the *output*. The algorithm, as a set of rules proceeding from inputs to output, must be precise and definite, with each successive step clearly determined. The concept of *computability* concerns those **objects** which may be **specified** in principle by computations." (las negrillas son nuestras) ([Soare, 1996], pág. 286)

Una *computación* es un proceso por el cual nosotros procedemos a partir de unos objetos iniciales, llamados *entradas*; de acuerdo a un conjunto fijo de reglas, llamadas un *programa*, *procedimiento* o *algoritmo*; a través de una serie de *pasos* y llegamos al final de estos pasos con un resultado final, llamado la *salida*. El algoritmo, como un conjunto de reglas de procedimiento desde las entradas hasta las salidas, debe ser preciso y definitivo, con cada pasa sucesivo claramente determinado. El concepto de computabilidad concierne con aquellos objetos los cuales pueden ser especificados en principio por computaciones.

La palabra "*object"* puede traducirse al español como "objeto" o "cosa", seleccionamos esta última debido a su

Es plausible interpretar la noción de computabilidad como una propiedad atribuible o no a cierta "clase" de objetos —se hablará entonces, de objetos computables y objetos no computables—. Es muy heterogénea[1] la clase de objetos que permite la pregunta por la computabilidad o no de sus miembros; por citar algunos ejemplos, es posible hablar de funciones computables o no computables (este es el objeto utilizado por la teoría de la computabilidad), de números computables o no computables (este fue el objeto seleccionado por Turing en su artículo [Turing, 1936-37]) y en una instancia de mayor cobertura, de procesos computables o no computables (objeto seleccionado por Penrose y tratado exhaustivamente en [Penrose, 1989] y [Penrose, 1994]). Una particularidad muy significativa de la propiedad de computabilidad, es su binariedad mutuamente excluyente, es decir un objeto es o no es computable, esta característica genera una partición en la clase de los objetos.

[1] A partir de este momento, la noción de objeto dará por supuesto que él pertenece a esta clase.

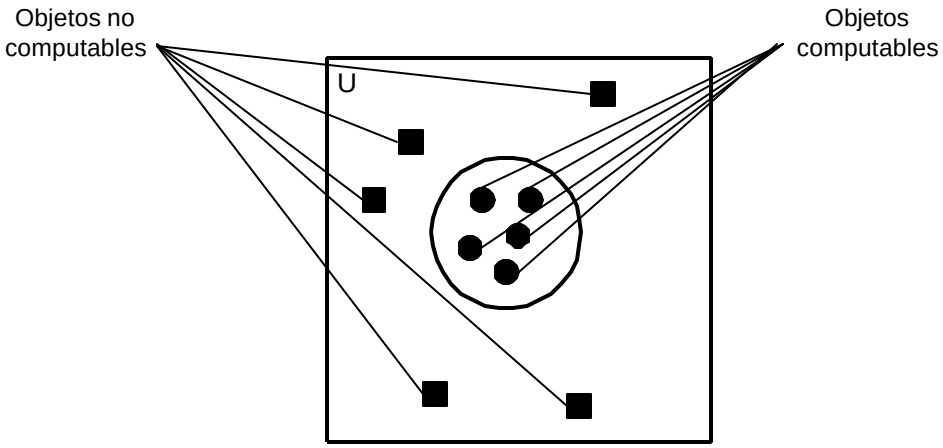


Figura 1: Partición de la clase de los objetos: Objetos computables y objetos no computables

¿Cómo identificar la propiedad de computabilidad o no computabilidad de un objeto?. Es necesario contar con un criterio de demarcación que construya la frontera (continua) entre los objetos computables y los que no lo son, esta frontera es la noción misma de computabilidad, pero no la definición informal presentada por Soare, sino una definición formal que permita distinguir con toda claridad que objeto es computable y cual no lo es. En la actualidad existen diferentes "versiones" para la definición formal de computabilidad —versiones que como es bien conocido, satisfacen la propiedad de ser coextensivas—; la versión con la cual se va a trabajar es la relacionada con las máquinas de Turing: un objeto es computable si es computable por una máquina de Turing.

La propiedad de computabilidad dota a los objetos (en principio) de una inteligibilidad completa, un objeto computable es un objeto conocido, es un objeto cuya aprehensión es plausible, es un objeto sintética y analíticamente descriptible, pero el precio que se debe pagar por este "dominio" del objeto es muy alto, un objeto computable es un objeto simple, es un objeto trivial (en el mismo sentido que las máquinas triviales de von Foerster [von Foerster-1984]).

En el mundo formal o en el mundo factual existen objetos que no son computables, por el lado formal se presentan por ejemplo: el problema de la parada de una máquina de Turing ([Turing, 1936-37]) o el problema de la teselación **(**Penrose, 1994]); por el lado factual, se mencionan procesos tales como: la morfogénesis ([von Bertalanffy-1968]) y la cognición ([Penrose, 1994]).

Es por esta clasificación binaria que se afirma que la noción de computabilidad actúa como un paradigma, es un filtro que clasifica los objeto en duplas, dominados y no dominados, simples y complejos, triviales y no triviales. Es frecuente que la ciencia realice grandes esfuerzos en "pulir" sus objetos para que crucen el filtro y se conviertan así, en objetos aprehensibles; esta es la versión del paradigma de la simplicidad observado desde la perspectiva de la computabilidad.

La definición formal de computabilidad, ha sido puesta en entredicho desde sus primeras apariciones. Con alguna frecuencia se han presentado personas que creen que la definición de computabilidad no es completa ni definitiva, personas que han intentado romper el paradigma de la computabilidad, por medio de definiciones más potentes de la misma. Se presenta apartes de la carta enviada por Alonso Church a Józes Pepis, con relación a la propuesta (implícita) de Pepis de una definición más poderosa de computabilidad. La carta cobra importancia en la medida que es escrita por Church, que como es sabido es el creador de una de las "versiones" formales de

computabilidad; por otra parte, la excelente explicación de Church de las consecuencias de contar con una noción más potente de computabilidad permite justificar el continuar con su búsqueda, aunque también ofrece argumentos bastante sólidos de la imposibilidad de encontrarla; pero más importante, es la posición de escepticismo adoptada por Church, muy diferente a la posición dogmática adoptada por algunos en la actualidad.

“Dear Mgr. [Monsignore] Pepis:

... In reply to your postal [card] I will say that I am very much interested in your results on general recursiveness, and hope that I may soon be able to see them in detail. In regard to your project to construct an example of a numerical function which is effectively calculable but not general recursive I must confess myself extremely skeptical - although this attitude is of course subject to the reservation that I may be induced to change my opinion after seeing your work.

I would say at the present time, however, that I have the impression that you do not fully appreciate the consequences which would follow from the construction of an effectively calculable non-recursive function.

For instance, I think I may assume that we are agreed that if a numerical function *f* is effectively calculable then for every positive integer *a* there must exist a positive integer *b* such that a valid proof can be given of the proposition *f(a) = b* (at least if we are not agreed on this then our ideas of effective calculability are so different as to leave no common ground for discussion). But it is proved in my paper in the American Journal of Mathematics that if the system of Principia Mathematica is omega-consistent, and if the numerical function *f* is not general recursive, then, whatever permissible choice is made of a formal definition of *f* within the system of Principia, there must exist a positive integer *a* such that for no positive integer *b* is the proposition *f(a) = b* provable within the system of Principia. Moreover this remains true if instead of the system of Principia we substitute any one of the extensions which have been proposed (e.g. allowing transfinite types), or any one the forms of the Zermelo set theory, or indeed any system of symbolic logic whatsoever which to my knowledge has ever been proposed.

Therefore to discover a function which was effectively calculable but no general recursive would imply discovery of an utterly new principle of logic, not only never before formulated, but never before actually used in a mathematical proof - since all extant mathematics is formalizable within the system of Principia, or at least within one of its known extension. Moreover this new principle of logic must be of so strange, and presumably complicated, a kind that its metamathematical expression as a rule of inference was not general recursive (for this reason, if such a proposal of a new principle of logic were ever actually made, I should be inclined to scrutinize the alleged effective applicability of the principle with considerable care).” (En: Revista Universidad EAFIT. No. 108. pág. 61 - 106.

[Sieg, 1997], pág. 175 - 176)[2]

Este es el contexto en el cual se construye la idea que conduce esta tesis: Presentar la posibilidad de ampliar la definición de computabilidad: Gráficamente:

[2]Algún lector podrá objetar que la propuesta de Pepis no es ampliar la noción de computabilidad, sino de refutar la tesis de Church-Turing. En algunos párrafos posteriores, se presenta la relación entre la tesis de Church-Turing y la propuesta de una nueva definición de computabilidad.

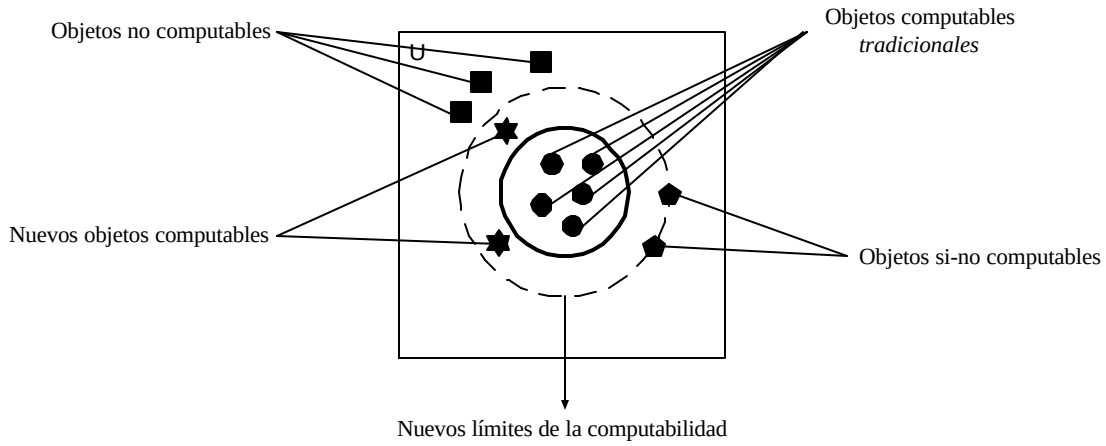


Figura 2: Ampliación definición de computabilidad

En la figura se observa que de nuevo existen objetos que no son computables (representados por un cuadrado), objetos computables tradicionales —es decir aquellos que son computables bajo la definición actual de computabilidad— (representados por un circulo) y objetos que bajo la definición actual de computabilidad no son computables, pero que bajo la ampliación de la definición sí lo son (representados por una estrella). Esta conversión de objetos no computables en objetos computables exige *a–priori* una jerarquía de la no computabilidad, es decir, deben existir diferentes *grados* de no computabilidad, de lo contrario, si todos los objetos no computables tienen el mismo *grado*, una vez se logre convertir uno de ellos en computable, se lograría convertirlos a todos (al momento de elaborar esta tesis, se tiene una idea de la existencia de estos grados de no computabilidad a partir de una bibliografía recién obtenida ([Rogers, 1992]), la cual una vez revisada permitirá ampliar esta idea).

Pero de mayor importancia, la discontinuidad presente en la frontera construida por la nueva definición de computabilidad —a diferencia de la continuidad bajo la definición actual—, representa el compromiso con la propuesta moriniana: la ratificación de la *complejidad*. Se admite las limitaciones y mutilaciones causadas por los métodos simplificadores y, se reconoce la necesidad de contar con nuevos *operadores—principios—definiciones* difusos para construir una imagen más fiel de lo real. Los objetos no computables son objetos complejos por naturaleza, incluso se puede afirmar que la complejidad inherente a ellos (por lo menos en muchos casos), es la causa de su no computabilidad. Características tales como recursividad organizacional, emergencia de propiedades, inconsistencia, indeterminismo, etc., hacen parte de sus cualidades y por extensión de los obstáculos epistemológicos que emergen en el intento de aprehenderlos. La propuesta de la complejidad, libera y amplia la lógica clásica para permitir que se incorporen nuevas categorías de verdad. En el contexto de la computabilidad se acepta y espera la nueva categoría de objetos sí-no computables (representados por un pentágono). Por supuesto, esto es inadmisible bajo los ojos consistentes, binarios de la ciencia actual, pero esta es precisamente la propuesta de Edgar Morin, en sus palabras:

"No se trata de retomar la ambición del pensamiento real simple de **controlar y dominar lo real**. Se trata de ejercitarse en un pensamiento capaz de tratar, de dialogar, de negociar, con lo real. " **(**as negrillas son nuestras) ([Morin, 1990], pág. 22).

Un magnifico ejemplo del fuerte tejido que existe entre la complejidad, la computabilidad y la ampliación de la computabilidad, lo ofrecen algunos trabajos realizados en construir modelos para

los sistemas vivos ([Rocha, 1994], [Kampis, 1992]). Kampis a partir del hecho de que la evolución es uno de las principales características de los sistemas vivos y, que esta evolución produce innovaciones en el sistema, las cuales aumentan la complejidad del mismo y; a partir de las limitaciones de los modelos computables actuales en donde es necesario conocer el futuro antes de que este pueda ser computado, es decir, una computación debe saber de antemano que va a computar, concluye acerca de la imposibilidad de utilizar la metáfora de la máquina computable para describir dichos sistemas. Además, dado que los sistemas vivos presentan la propiedad de auto–modificar su comportamiento, propiedad que escapa a ser modelada por reglas *a–priori*, es necesario contar con nuevos modelos de computabilidad, que sean capaces de modificar su comportamiento en tiempo de ejecución.

Por otra parte, con base en los comentarios realizados por Soare, en relación con la aceptación de la tesis de Church–Turing:

"This may be viewed as roughly analogous to Euclidean geometry or Newtonian physics capturing a large part of everyday geometry or physics, but not necessarily all *conceivable* parts. Here, Turing has captured the notion of a function computable by a mechanical procedure, and as yet there is no evidence for any kind of computability which is *not* included under this concept. **If it existed, such evidence would not affect Turing's Thesis about mechanical computability any more than hyperbolic geometry or Einsteinian physics refutes the laws of Euclidean geometry or Newtonian physics. Each simple describes a different part of the universe**.

... Some have cast doubt on Turing's Thesis on the grounds that there might be physical or biological processes which may processes, say, the characteristic function of the halting problem. **It is possible that these may exist** (although there is presently no evidence) **but if so, this will have absolutely *no effect* on Turing's Thesis because they will not be algorithmic or mechanical procedures as required in §2.1 [3] and in Turing's Thesis.**" ([Soare, 1996], pág. 294 - 295; las negrillas son nuestras)

Se evita incursionar en la relación entre la amplitud de la computabilidad y la tesis de Church–Turing. Esto no quiere decir que se ignore el campo problemático planteado por ella. Una vez ampliada la definición de computabilidad, sería necesario regresar a la tesis de Church–Turing y reflexionarla con respecto a la nueva noción, quizás para construir una tesis ampliada (como lo es la relación entre la física de Newton y la física de Einsten), quizás para construir una tesis alternativa (como lo es la relación entre la geometría euclidiana y la geometría hiperbólica).

Hecha la apuesta por la posibilidad de ampliación de la definición de computabilidad y por extensión directa, el fortalecimiento de los aparatos de captura de lo real, surge las siguientes preguntas: ¿Cuál es la posibilidad de repetir este proceso? ; ¿Se puede repetir *ad–infinitum* o tiene límite?, ¿Cuáles son los alcances de estas múltiples ampliaciones?. En términos generales la pregunta es por los límites de la ciencia, con la complejidad a bordo por supuesto y con la noción de objeto computable —bajo nuevas y más potentes definiciones de computabilidad— como su elemento base. Se defiende la infinitud del proceso, es decir existe optimismo en la capacidad humana para aumentar sus aparatos de cognición, pero aunque el proceso es infinito tiende a una asíntota insuperable, es decir se acepta la incompletitud final del proceso. Este es un juego contra la naturaleza que no se puede ganar. He aquí el sentido trágico de esta aventura llamada ciencia, pero el mismo constituye su esencia.

[3] Los requerimientos a los que se refiere Soare, fueron presentados al comienzo de esta sección.

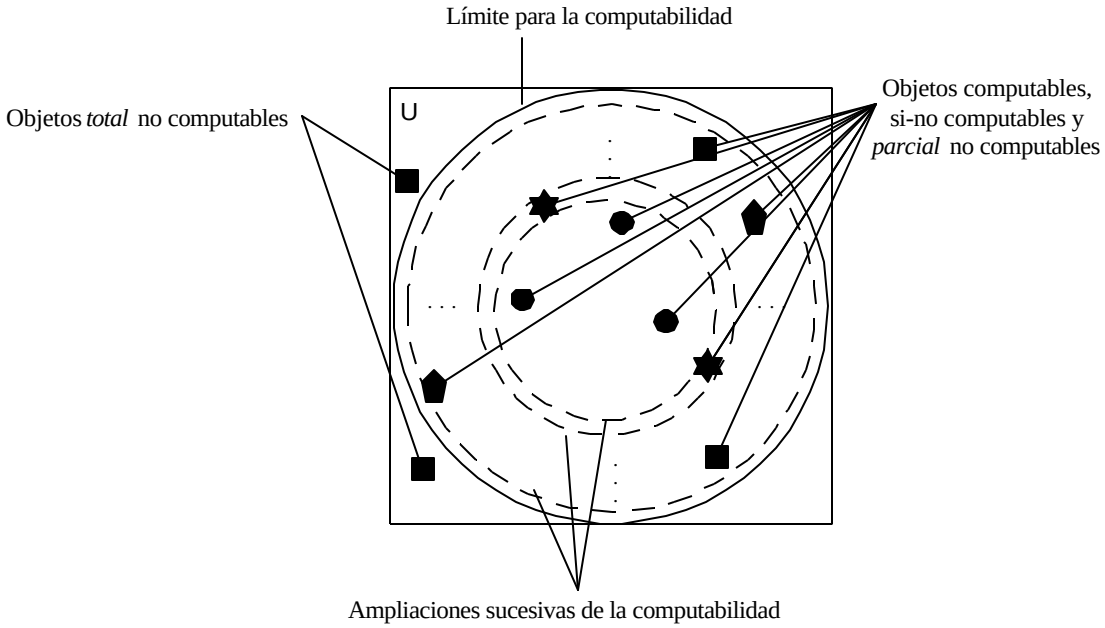


Figura 3: Asíntota para las ampliaciones de la definición de computabilidad

2 ¿CÓMO SE GESTÓ LA IDEA DE LA MÁQUINA DE TURING DINÁMICA?

La primera vez que el autor observó el término “máquina de Turing” fue en el texto: “El robot Karel” ([Pattis-1985])—primer texto que se utilizó para aprender a programar— durante el primer semestre cursado de la carrera de ingeniería de sistemas, en el año de 1988. El último problema del libro —que por supuesto no se realizó— hacia mención a las máquinas de Turing[4]. Tal como el problema lo menciona, el autor tuvo la oportunidad de recibir un curso de teoría de la computabilidad, este curso se llamó matemáticas especiales III, fue recibido en el semestre 1989-2 y ofrecido por el profesor Raúl Gómez. Se considera este curso como la primera relación del autor con las máquinas de Turing y de allí nació el gusto por el tema. Por algunos años el tema permaneció olvidado (incluso, aunque se realizó la tesis de pregrado asesorados por el profesor Raúl y se trabajó en temas de informática teórica, las máquinas de Turing no fueron mencionadas.). En el semestre 1994-2 el autor se vinculó a la universidad EAFIT como profesor y el primer curso que ofreció fue el de matemáticas especiales III, el tema de las máquinas de Turing se preparó a partir de las notas de clase y del libro de Xavier Caicedo[Caicedo, 1990]. Para semestres posteriores se adquirió alguna bibliografía adicional, se menciona entre otros: [Hopcroft, 1984], [Kleene, 1974], [Sáez-Fernández, 1987], [Minsky, 1967], [Hermes, 1969].

Para el año de 1996 era evidente que faltaba un texto en la bibliografía obtenida hasta el momento: el artículo original de Turing donde presentaba sus máquinas —artículo que se había visto citado muchas veces— ;además los motivos expuestos por Eco([Eco, 1977]) acerca de la conveniencia de leer de las fuentes originales, reforzaban el deseo de obtener el texto original. Por medio de uno de nuestros compañeros de colegio, que se encontraba en la ciudad de Sao Paulo, se pudo obtener el artículo buscado. La posibilidad de disfrutar del artículo original de Turing causó mucha expectativa. El artículo se recibió por la tarde y esa misma noche se procedió a leerlo.

Se transcribe textualmente una parte de la descripción (informal) realizada por Turing acerca de las *computing machines* (hoy en día llamadas Máquina de Turing):

“... We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions q_1, q_2, ..., q_R which will be called “*m*- configurations”. The machine is supplied with a “tape” (the analogue of paper) running through it, and divided into sections (called “squares”) each capable of bearing a “symbol”. At any moment there is just one square, say the *r*-th, bearing the symbols S(r) which is “in the machine”. We may call this square the “scanned square”. The symbol on the scanned square may be called the “scanned symbol”. The “scanned symbol” is the only one of the which the machine is, so to speak, “directly aware”. However, by altering its *m*-configuration the machine can effectively remember some of the symbols which it has “seen” (scanned) previously. The possible behavior of the machine at any moment is determined by the *m*-configuration q_n and the scanned symbol S(r). This pair q_n, S(r) will be called the “configuration”: thus the configuration determines the possible behavior of the machine. In some of the configurations in which the scanned square is blank (*i.e.* bears no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol. The machine may also

[4] El problema es el siguiente:
“Si usted disfruta de las ciencias de la computación y llega a tomar un curso sobre teoría de la computabilidad, recordará con cariño los días que paso programando en Karel y tratará de resolver el siguiente problema: pruebe que Karel, incluso sin el auxilio de zumbadores, es equivalente a una máquina de Turing. **Clave**: utilice la equivalencia entre las máquinas de Turing y un autómata de dos contadores. Recuerde que las instrucciones de Karel no son mutuamente recursivas, así que la información de estado debe ser codificada en alguna otra forma.”([Pattis-1985] pág. 147)

change the square which is being scanned, but only by shifting it one place to right or left. In addition to any of these operations the *m*-configuration may be changed. ... " (**[Turing, 1936-37]**, pág. 231)

Esta es la descripción realizada por Turing acerca de la máquina que lleva su nombre, esta descripción no diferenciaba ni ampliaba para nada la descripción que se conocía, excepto por la última frase: ***In addition to any of these operations the m-configuration may be changed***. Debido a la lectura rápida que se realizó en un primer momento, no se correlacionó el término *m-configuration* con el término de *estado*, sino que se asoció al conjunto de instrucciones de la máquina. Es imposible desentrañar de donde provino el error, probablemente se pensó que la relación: *m-configuration* ≡ *machine-configuration* ≡ *descripción-máquina* ≡ *instrucciones-máquina* era correcta; lo cierto era que la lectura bajo la noción erronea de *m-configuration* era sorprende: **!el conjunto de instrucciones de la máquina puede ser cambiado!**. Esto gatillo inmediatamente la idea de construir una máquina de Turing con esta propiedad y se decidió llamarla: máquina de Turing dinámica (MTD).

En una de las lecturas posteriores del artículo de Turing —realizadas con mayor tranquilidad—se rectificó el error, pero de alguna forma "ya era demasiado tarde", la intuición de construir la máquina de Turing Dinámica ya había comenzado su misterioso y sorprendente camino en nuestra mente; se revisó la intuición a luz del error que se había detectado y no la modificada para nada, se recordó que en la historia de la ciencia frecuentemente se ha partido de ideas erróneas y obtenido buenos resultados y se decidió continuar adelante.

En los cursos actuales que se ofrecen de metodología de la investigación, se relata esta historia a los estudiantes, intentando explicar por medio de un ejemplo —nuestro ejemplo— de donde surgen las intuiciones que después adquieren el *status* de hipótesis. Decir que nuestra hipótesis surgió a partir de mal interpretar en lectura inicial un concepto (el concepto de *m-configuration*) es una forma muy ingenua de explicación. La mala interpretación que se hizo "gatillo" el problema, pero de ahí a decir que el problema surgió de esta equivocada interpretación, es desconocer el pasado. El autor esta convencido que la idea o por lo menos la posibilidad de la idea habitaba en él, incubada a partir de su forma de observar y relacionarse con el mundo. Se imagina la idea en el inconsciente a la espera que algún suceso le abra la puerta.

3 ¿QUÉ ES UNA MÁQUINA DE TURING DINÁMICA?

Una MTD es una MT en la cual es posible modificar su conjunto de instrucciones. Por supuesto, si se construye una máquina de Turing M_x, el diseñador puede modificar algunas instrucciones y construir una nueva máquina de Turing $M_{x'}$. Es decir, en principio siempre es posible modificar una MT para obtener una nueva MT. Entonces, ¿qué significa que en una MTD es posible modificar su conjunto de instrucciones?

La característica subyacente a la modificación de instrucciones en una MTD, radica en el agente ejecutor de esta modificación: ella misma. Es decir, la MTD está dotada de una propiedad de autorreferencia que le permite modificar su propia descripción-comportamiento.

Pero es necesario realizar un análisis más cuidadoso de lo anterior, la descripción de una máquina de Turing es un conjunto de símbolos escritos en un papel, cómo es posible que estos símbolos se modifiquen ellos mismo, para poder hablar de una MTD, sería como si los símbolos pudieran coger un lápiz y un borrador y escribir de nuevo sobre el papel y esto por supuesto es imposible. Es en este punto donde adquiere relevancia la máquina universal de Turing (MUT).

Pero, aunque dado que la MUT es una MT que ejecuta máquinas de Turing, ¿cómo es posible hablar de una MT que cuando sea ejecutada por la MUT modifique su descripción, para ser llamada una MTD?. Para contestar esta pregunta, es necesario entrar un poco más en detalle en la relación entre la MUT y la MT que está ejecuta.

La MUT recibe como parámetro la MT, codificada en un código especial construido por Turing para ello (ver Anexo No.1). El comportamiento de una MTU puede ser esquematizado así:

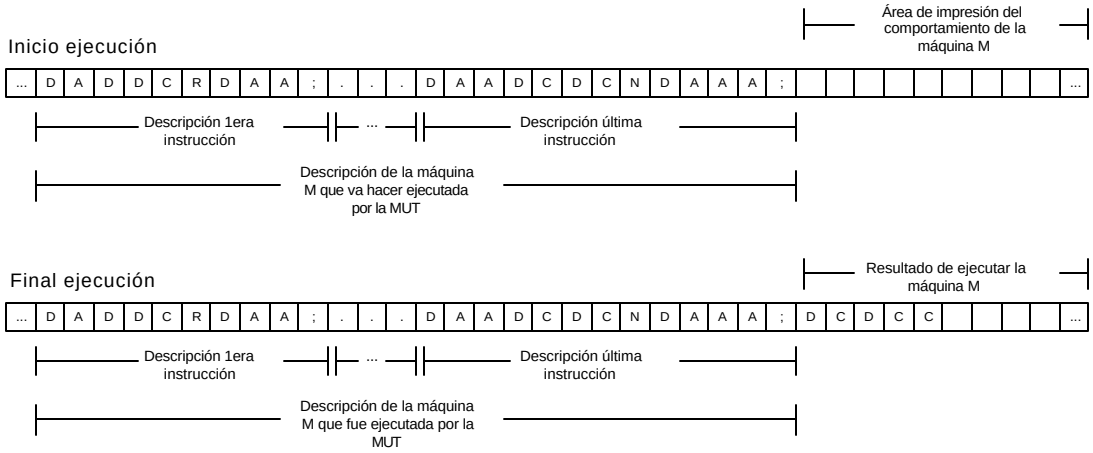


Figura 4: Ejecución de una MT por medio de la MUT

Para efectos de ilustración, no interesa las instrucciones que componen la máquina M, ni sus resultados. Lo que sí es importante observar es que la descripción de la máquina M no se modifica durante su ejecución por medio de la MUT. La única "sección" que se modifica es la correspondiente al área donde se imprime el comportamiento de la máquina M.

Cuando se habla de una MTD, es decir de una MT que sea capaz de modificar su descripción-comportamiento, es necesario añadir que este cambio en la descripción-comportamiento ocurre durante su ejecución por la MTU, es decir, se habla de un cambio en la descripción-comportamiento en tiempo real. Gráficamente:

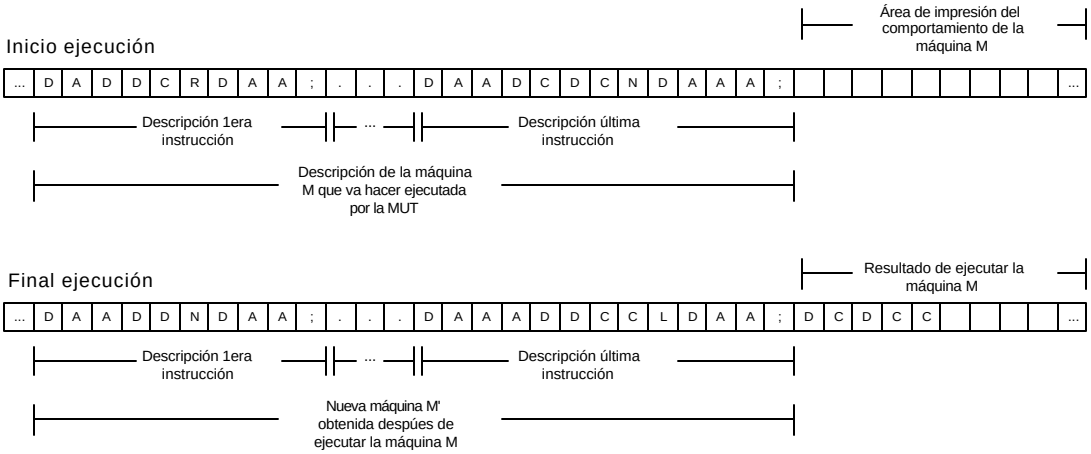


Figura 5: Ejecución de una MTD por medio de la MUT

Es decir la máquina M se auto-modifica y se convierte en una máquina M'. Para poder realizar esto, la máquina M debe contar con instrucciones especiales que le indiquen como modificarse. De la Figura 5: Ejecución de una MTD por medio de la MUT, se puede observar que dichas instrucciones deben poder escribir en la sección correspondiente a la descripción de la máquina M, adicionalmente se observa que cualquier instrucción puede modificarse y en particular cualquier parte de la instrucción (estado-actual, símbolo-leer, símbolo-escribir, movimiento, estado-siguiente).

4 ¿QUÉ SIGNIFICA QUE UNA MÁQUINA DE TURING PUEDA MODIFICAR SU DESCRIPCIÓN-COMPORTAMIENTO?

La relación que se considera más importante entre una MT y MTD es la relación de potencia. ¿Es la MTD menos potente, igual de potente o más potente que una MT?. A manera informal se presentan las siguientes consideraciones:

¿Es la MTD menos potente que la MT?. Esta alternativa puede ser descartada trivialmente debido a que la "definición" de la MTD indica que ésta es una MT extendida, es decir la MTD tiene una propiedad que no tiene la MT.

¿Es la MTD igual de potente que la MT?. Esta alternativa es evidente dado que es fácil observar que cualquier MT es una MTD, es decir cualquier objeto que puede ser computado por una MT, puede ser computado por una MTD, ya que la construcción de la MTD consistiría en la misma MT que se desea simular.

En último lugar, ¿es la MTD más potente que la MT?. El lector debe considerar muy cuidadosamente lo que esto significa, existirían objetos que podrían ser computados por un MTD, pero no podrían serlo por una MT. Estos objetos serían la prueba de una nueva y más potente definición de computabilidad.

Se analiza con mayor detalle las diferencias entre la MTD y la MT en aras de comprender la relación de potencia entre las mismas. Dada la característica de auto-modificación de la MTD, está puede ser cualquiera MT en algún momento **discreto**. Además a partir de que para cualquier máquina de Turing existe un número de descripción que la identifica (ver Anexo No. 1), se puede afirmar que la MTD puede comenzar siendo la n-ésima MT, pero después de algunas instrucciones ser la n'-ésima MT y después de algunas instrucciones ser la n''-ésima MT. La palabra discreto resaltada al comienzo de este párrafo es de suma importancia, si se "fotografía" la ejecución de una MTD, en cada instante se puede observar una MT diferente[5]. Pero que ocurre si no se "fotografía" la descripción de la MTD durante su ejecución por la MUT, sino que más bien se "filma", ¿qué se observa?

Por otra parte una MT es una máquina con características de finitud muy especificas, su alfabeto es finito, su conjunto de estados es finito y su conjunto de instrucciones es finito[6]. No sería difícil concebir una MTD que modifique el estado q_n de alguna instrucción por el estado q_{n+1}, tampoco sería difícil concebir una MTD que modifique el símbolo s_n de alguna instrucción por el símbolo s_{n+1}. ¿Afectan estas modificaciones las restricciones de finitud mencionadas con anterioridad?. De nuevo se recurre a la distinción entre "fotografiar" o "filmar" la descripción de la MTD durante su ejecución por la MUT. Si se "fotografía", la finitud del alfabeto, del conjunto de estados y por consiguiente del conjunto de instrucciones, se preserva, pero si se "filma" un tiempo suficientemente largo ¿qué ocurre con estas finitudes?

El número de máquinas de Turing que se pueden construir es infinito pero enumerable, es decir $\aleph_0$. Si una MTD puede ser cualquier MT en algún instante, la demostración de mayor potencia de

[5] Técnicamente esto no exacto ya que toma más de un paso de ejecución el modificar alguna instrucción.

[6] La finitud del conjunto de instrucciones es una consecuencia de la finitud del alfabeto y del conjunto de estados, tal como se menciona en el anexo No. 1.

la MTD sobre la MT se realizaría demostrando un teorema con la siguiente forma: $\bigvee_{i=0}^{\aleph_0} MT_i \phi\ MT_C$, es decir, sería necesario demostrar que la unión enumerable de todas las máquinas de Turing, es más potente que cualquiera de ellas.

5 ¿ES POSIBLE FORMALIZAR LA IDEA DE LA MÁQUINA DE TURING DINÁMICA?

5.1 Introducción:

Para la MTD, se desea formalizar su propiedad de autorreferencia, propiedad que le permite modificar su descripción-comportamiento en tiempo de ejecución, es decir, en el momento en que es ejecutada por la MUT.

La noción base en esta formalización, es la noción de *m-función*. Una m-función puede ser vista como una MT con parámetros[7]. En este contexto los términos instrucción y m-función serán usados intercambiablemente.

Se espera que una MTD este formada por un conjunto de m-funciones que indican su comportamiento *tradicional* y por un conjunto de m-funciones que representan su capacidad de autorreferencia.

[7] Se recomienda al lector leer los anexos No. 1 y No. 2 para familiarizarse con el uso de las m–ciones.

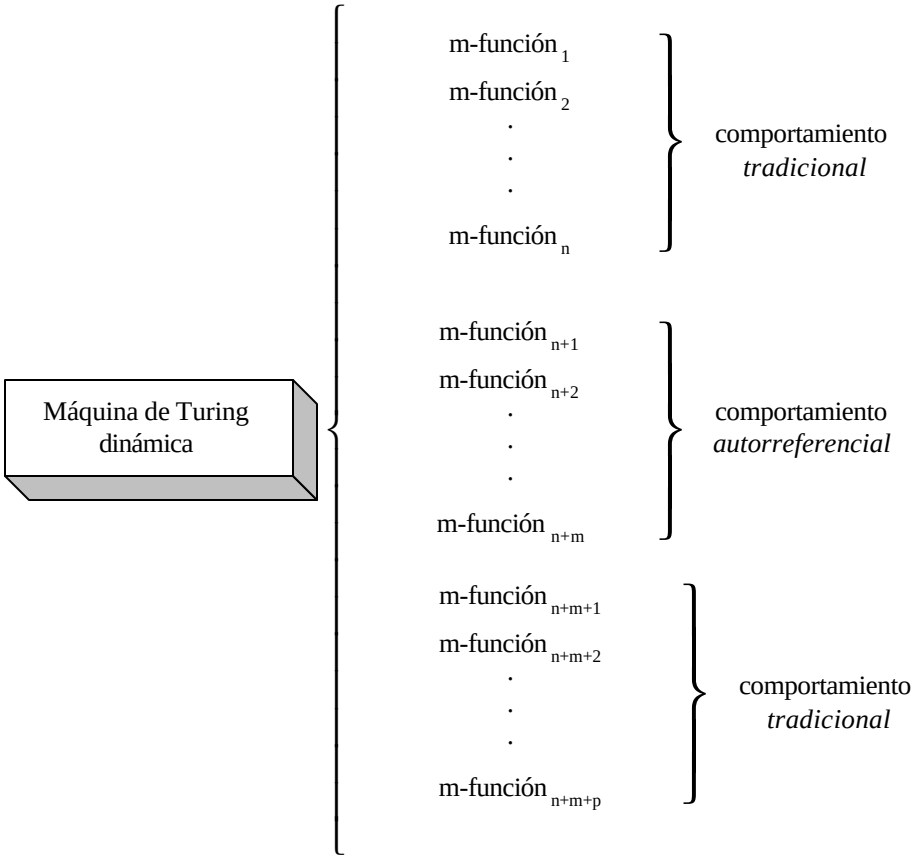


Figura 6: m–funciones tradicionales y m–funciones autorreferenciales en la MTD

Este capítulo está dedicado a ampliar la descripción del conjunto de instrucciones que implementan el comportamiento autorreferencial. Con respecto a las instrucciones que realizan el comportamiento tradicional, se afirma que estas no "interesan", dado que tal como lo ilustra la figura, éstas son independientes de las instrucciones autorreferenciales[8].

La característica inicial de las instrucciones autorreferenciales es que estas sean independientes de las instrucciones de comportamiento. Es decir, se espera poder modificar cualquier instrucción de comportamiento; en este punto se menciona que la posibilidad de autorreferencia de segundo grado, es decir, la posibilidad de modificar las instrucciones de autorreferencia por ellas mismas, no es contemplada.

Para lograr la independencia en la descripción de las instrucciones de autorreferencia, de la *instrucción original* y la *instrucción final*, es decir entre la instrucción que se va a cambiar y de la instrucción por la cual se va a cambiar, se decidió construir una m-función que recibiera estas dos instrucciones como parámetro.

[8] En realidad sí existe relación entre unas y otras. Las instrucciones autorreferenciales modifican a las instrucciones de comportamiento, pero las instrucciones autorreferenciales reciben por parámetro las instrucciones que van a modificar, como será descrito posteriormente.

Todo el comportamiento de autorreferencia está implementado en esta m-función llamada **cambiarInstrucción**, la cual a su vez está implementada con base en otras m-funciones.

5.2 Definición m–función cambiarInstrucción

5.2.1 Encabezado

La m-función **cambiarInstrucción** tiene como parámetros la instrucción que se va a cambiar —*instrucciónOriginal*— y la instrucción por la cual se va a cambiar —*instrucciónFinal*—. Es decir: **cambiarInstrucción**(*instrucciónOriginal*, *instrucciónFinal*)

De acuerdo con la "notación tradicional" usada para escribir las instrucciones, los parámetros de la m-función **cambiarInstrucción** son de la forma:

cambiarInstrucción(q_i, s_j, s_k, m, q_l, q'_i, s'_j, s'_k, m', q'_l) donde:
"q_i, s_j, s_k, m, q_l" representa la *instrucciónOriginal* y "q'_i, s'_j, s'_k, m', q'_l" representa la *instrucciónFinal*.

De acuerdo con la codificación de instrucciones empleada por Turing, la codificación para la *instrucciónOriginal* es de la forma:

Codificación	D A 1 o más veces	D C 0 o más veces	D C 0 o más veces	L o R o N	D A 1 o más veces
Símbolo que representa	q_i	s_j	s_k	m	q_l

Similarmente, la codificación de la *instrucciónFinal* es de la forma:

Codificación	D A 1 o más veces	D C 0 o más veces	D C 0 o más veces	L o R o N	D A 1 o más veces
Símbolo que representa	q'_i	s'_j	s'_k	m'	q'_l

Aunque la codificación anterior sugiere la necesidad de una notación muy particular para los parámetros de la m-función **cambiarInstrucción**, para efectos de este desarrollo se van a representar por **cambiarInstrucción**(*instrucciónOriginal*, *instrucciónFinal*) donde los parámetros *instrucciónOriginal* e *instrucciónFinal* representan por implícito, las cadenas de símbolos que corresponden en la notación empleada por Turing de la instrucción original y la instrucción final respectivamente.

La forma de recibir los parámetros de la m-función **cambiarInstrucción** permite la modificación de la instrucción original, sea a nivel "atómico", es decir, cada una de las partes de la instrucción original (estado-actual, símbolo-leer, símbolo-escribir, movimiento, estado-siguiente) puede ser cambiada independientemente —tal como se menciono en la sección: ¿Qué es una máquina de Turing dinámica?—. Para observar esto, se puede pensar en los parámetros de la m-función **cambiarInstrucción** como:

instrucciónOriginal	***instrucciónFinal***	***comentario***
"q_i, s_j, s_k, m, q_l"	"q'_i, s_j, s_k, m, q_l"	Se modifica el *estado-actual*
"q_i, s_j, s_k, m, q_l"	"q_i, s'_j, s_k, m, q_l"	Se modifica el *símbolo-leer*
"q_i, s_j, s_k, m, q_l"	"q_i, s_j, s'_k, m, q_l"	Se modifica el *símbolo-escribir*
"q_i, s_j, s_k, m, q_l"	"q_i, s_j, s_k, m', q_l"	Se modifica el *movimiento*
"q_i, s_j, s_k, m, q_l"	"q_i, s_j, s_k, m, q'_l"	Se modifica el *estado-siguiente*

Por otra parte, la notación empleada posibilita la modificación de cualquier número de elementos constituyentes de la instrucción original simultáneamente:

instrucciónOriginal	***instrucciónFinal***	***comentario***
“q_i, s_j, s_k, m, q_l”	“q'_i, s_j, s_k, m, q'_l”	Se modifican el *estado-actual* y el *estado-siguiente*
“q_i, s_j, s_k, m, q_l”	“q'_i, s'_j, s'_k, m, q_l”	Se modifica el *estado-actual*, el *símbolo-leer* y el *símbolo-escribir.*
“q_i, s_j, s_k, m, q_l”	“q'_i, s'_i, s'_k, m’, q'_l”	Se modifican todos los elementos de la instrucción-original

5.2.2 Cuerpo

Antes de realizar la descripción de la m-función **cambiarInstrucción**(*instrucciónOriginal*, *instrucciónFinal*) en la metodología y notación empleada para describir las m-funciones[9], se presenta su comportamiento a manera de algoritmo. En esta presentación se suprimió el parámetro de retorno que debe llevar toda m–función, es decir el estado al cual retorna una vez finalizado su comportamiento.

cambiarInstrucción(instrucciónOriginal, instrucciónFinal)
{
(1) **guardarPosiciónInicial()**
(2) **imprimirInstrucción**(*instrucciónOriginal*)
(3) **marcarInstrucción**(*o*)
por cada *instrucción*$_i$
(4) **marcarInstrucción**(*i*)
(5) **compararInstrucciones**(*i*, o)
si *instrucción*$_i$ = *instrucciónOriginal* entonces
(6) **imprimirInstrucción**(*instrucciónFinal*)
(7) **marcarInstrucción**(*f*)
(8) **reemplazarInstrucción**(*i*, *f*)
(9) **eliminarInstrucción**(*f*)
(10) **eliminarInstrucción**(*o*)
(11) **eliminarMarca**(i)
(12) **restablecerPosiciónInicial**()
return
sino
(13) **eliminarMarca**(i)
fin no
fin si
fin por cada
(14) **eliminarInstrucción**(*o*)
(15) **restablecerPosiciónInicial**()
}

Como ilustra el algoritmo anterior la m-función **cambiarInstrucción** está compuesta de varias m-funciones.

5.2.3 Descripción del comportamiento de la m-función cambiarInstrucción

El procedimiento empleado para realizar la descripción informal del comportamiento de la m-función **cambiarInstrucción** será a partir de un ejemplo que hará referencia a los aspectos fundamentales. Este procedimiento se debe a que por el momento no interesa ilustrar todos los detalles del comportamiento de la m–función **cambiarInstrucción** y por otra parte a que ésta está compuesta por un conjunto de m-funciones muy simples, en donde el nombre de las mismas refleja su comportamiento.

Sea MTD la máquina de Turing dinámica formada por las siguientes instrucciones:
Instrucciones tradicionales:
i_1: q_1 0 N q_1
i_2: q_1 0 0 R q_2

[9] Esta metodología y notación son las descritas en el anexo No. 2.

Instrucciones de autorreferencia:

cambiarInstrucción $\left(\underbrace{q_1 00 R q_2}_{\text{instrucción original}} , \underbrace{q_1 01 R q_2}_{\text{instrucción final}} \right)$

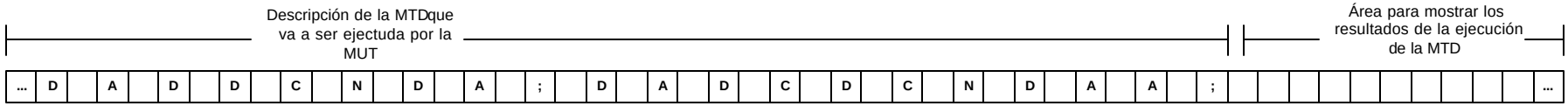
En este caso, en la instrucción i2 se va a modificar el símbolo-escribir ‘0’ por el símbolo-escribir ‘1’.
La codificación para las instrucciones es:
i_1: D A D D C N D A
i_2: D A D C D C R D A A

Con lo cual, la m-función **cambiarInstrucción** toma la forma:

cambiarInstrucción $(\underbrace{DADCDCRDAA}_{instruccionOriginal}, \underbrace{DADCDCCRDAA}_{instruccionFinal})$

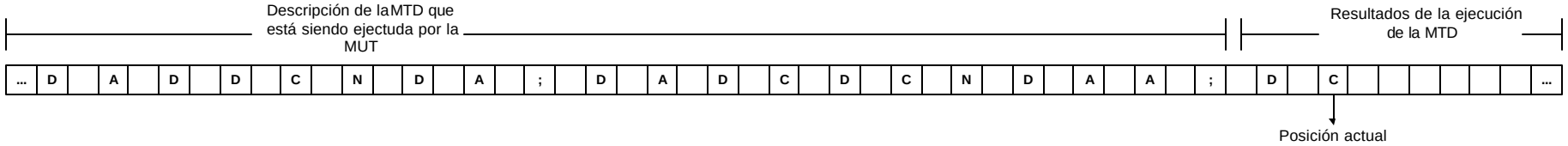
Al inicio de la ejecución de la MTD por la MUT se tiene:

Inicio



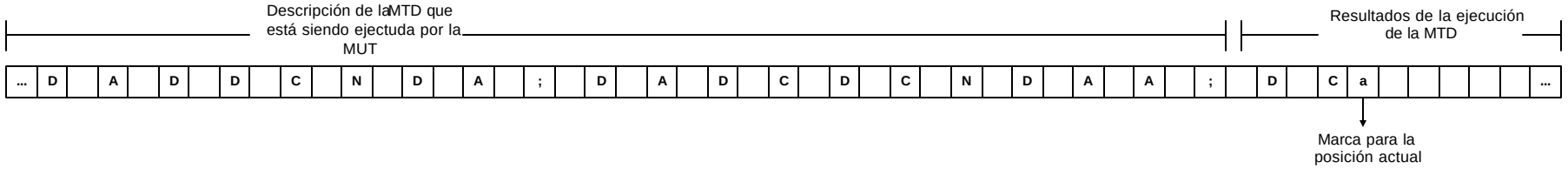
Después de ejecutar las instrucciones i_1 e i_2, se obtiene:

Ejecución instrucciones i_1 e i_2



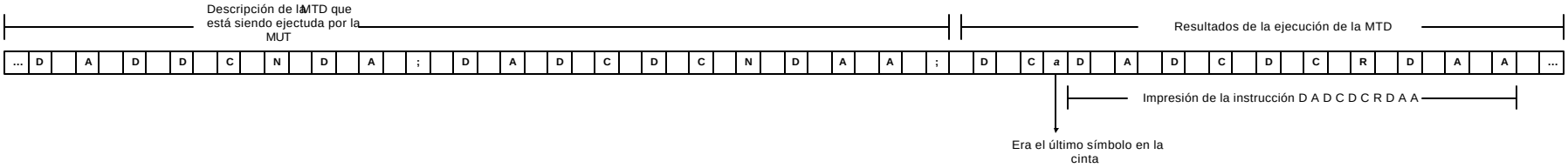
En este punto comienza la ejecución de la m-función **cambiarInstrucción**($\underbrace{D\ A\ D\ C\ D\ C\ R\ D\ A\ A}_{instruccionOriginal}$, $\underbrace{D\ A\ D\ C\ D\ C\ C\ R\ D\ A\ A}_{instruccionFinal}$). Inicialmente se ejecuta la línea (1) **guardarPosiciónInicial()**, en donde la m-función guardarPosiciónInicial() marca la posición actual con el símbolo ‘a’, entonces se obtiene:

Ejecución de la m-función **guardarPosiciónInicial**()



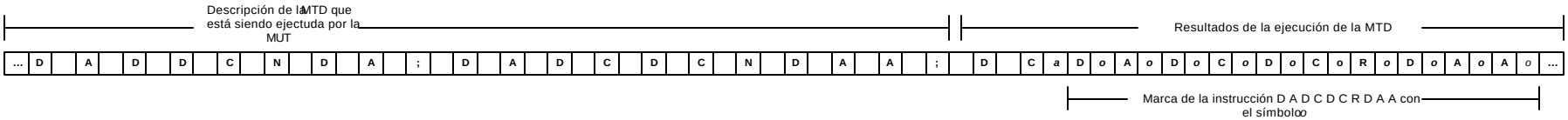
Luego se ejecuta la línea (2) m-función **imprimirInstrucción**(*instrucciónOriginal*) que para el ejemplo consiste en **imprimirInstrucción**(*D A D C D C R D A A*); esta m-función imprime la instrucción que recibe como parámetro después del último símbolo en el área de resultados, con lo cual se obtiene:

Ejecución de la m-función **imprimirInstrucción**(*D A D C D C R D A A*)



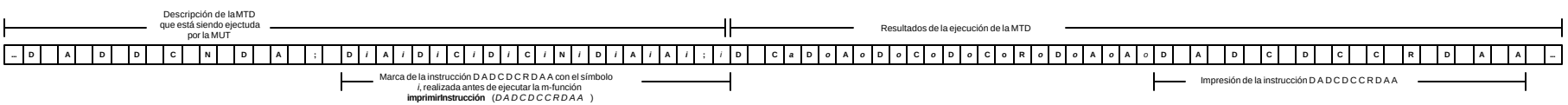
La línea (3) **marcarInstrucción**(*o*), marca la instrucción a la derecha de la posición actual con el símbolo *o*. Se obtiene:

Ejecución de la m-función **marcarInstruccción**(*o*)

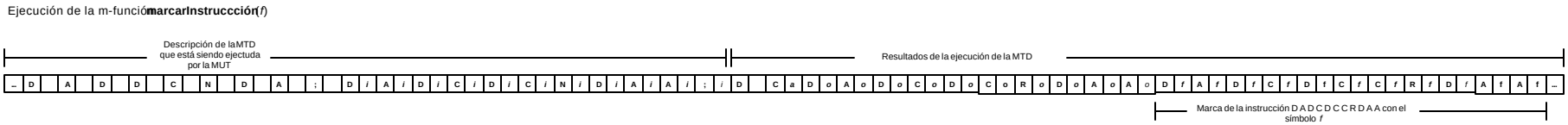


Las líneas (4) a (13) están dentro de un ciclo que itera una vez por cada instrucción de la MTD. Comenzando de izquierda a derecha. Inicialmente la línea (4) **marcarInstrucción**(*i*), marca con el símbolo *i* la primera instrucción, para el ejemplo es la instrucción D A D D C N D A; la línea (5) **compararInstrucciones**(*i*, o), compara las instrucciones marcadas con los símbolos *i* y *o*; en este caso las dos instrucciones son diferentes por lo que las líneas (6) a (12) no son ejecutadas y es ejecutada la línea (13) **eliminarMarca**(i) que elimina la marca *i* de la primera instrucción. Se realiza una nueva iteración del ciclo, en este caso la instrucción D A D C D C R D A A es marcada con el símbolo *i*. Ahora las instrucciones marcadas con los símbolos *i* y *o* son iguales, lo cual produce la ejecución de las líneas (6) a (12). Inicialmente la línea (6) **imprimirInstrucción**(*instrucciónFinal*) que para el ejemplo es **imprimirInstrucción**(*D A D C D C C N D A A*) imprime la instrucción que recibe como parámetro después del último símbolo en el área de resultados. Entonces:

Ejecución de la m-función **imprimirInstruccción**(*D A D C D C C R D A A*)

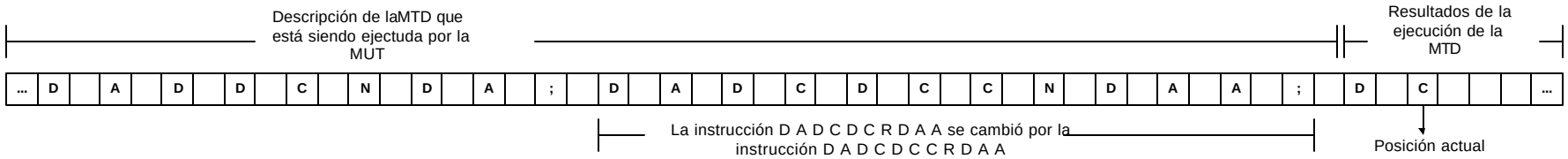


La línea (7) **marcarInstrucción**(*f*), marca con el símbolo *f* la instrucción a la derecha de la posición actual. Se obtiene:



La línea (8) **reemplazarInstrucción**(*i*, *f*), reemplaza la instrucción marcada con el símbolo i (instrucción original), por la instrucción marcada con el símbolo *f* (instrucción final). La línea (9) **eliminarInstrucción**(*f*) elimina la instrucción marcada con el símbolo *f* (instrucción final). La línea (10) **eliminarInstrucción**(*o*) elimina la instrucción marcada con el símbolo *o* (instrucción original). La línea (11) **eliminarMarca**(i), elimina la marca *i*. Finalmente la línea (12) **restablecerPosiciónInicial**() restablece la posición de la máquina al símbolo marcado con el símbolo *a* (símbolo en la cual estaba la máquina antes de ejecutar la m-función **cambiarInstrucción**) y elimina la marca *a.*

Ejecución de las m-funciones **reemplazarInstrucción**(*D A D C D C R D A A, D A D C D C C R D A A*); **eliminarInstrucción**(*f*); **eliminarInstrucción**(*o*); **eliminarMarca**(**i**); **reestablecerPosiciónInicial**(*S*)



Las líneas (14) **eliminarInstrucción**(*o*) y (15) **restablecerPosiciónInicial**(*S*) son ejecutadas en el caso de que la *instrucciónOriginal* recibida como parámetro, no corresponda a ninguna de las instrucciones de la MTD.

5.2.4 ¿Constructibilidad o no constructibilidad de la m-función cambiarInstrucción?

El título de este capítulo "¿Es posible formalizar la idea de la máquina de Turing dinámica?" refleja la incertidumbre acerca de la posibilidad de construir una MTD. De acuerdo a lo observado en las secciones precedentes, la posibilidad o imposibilidad de construir una MTD se reduce a la posibilidad o imposibilidad de especificar la m-función **cambiarInstrucción**. Es decir, el núcleo de la MTD es la m-función **cambiarInstrucción.**

La m-función **cambiarInstrucción** está compuesta de ocho m–funciones, a saber:

1. **guardarPosiciónInicial**()
2. **imprimirInstrucción**(*instrucción*)
3. **marcarInstrucción**(*marca*)
4. **compararInstrucciones**(*marca*$_1$, *marca*$_2$)
5. **reemplazarInstrucción**(*marca*$_1$, *marca*$_2$)
6. **eliminarInstrucción**(*marca*)
7. **eliminarMarca**(*marca*)
8. **restablecerPosiciónInicial**()

Con base en la simplicidad del comportamiento de cada una de estas m-funciones, se puede prever que su codificación no es excesivamente compleja, entonces porque suponer que es imposible. A manera de ejemplo se describen las m-funciones **imprimirInstrucción** y **marcarInstrucción**, adicionalmente se describe la m-función $\mathbf{PE_n}(S, \alpha_1, \alpha_2, ..., \alpha_n)$ que es usada por la m-función **imprimirInstrucción**[10].

5.2.4.1 Descripción m-función $PE_n(S, a_1, a_2, ..., a_n)$

5.2.4.1.1 Explicación

A partir de las m-funciones $\mathbf{PE}(\mathbf{S}, \beta)$ y $\mathbf{PE_2}(\mathbf{S}, \alpha, \beta)$ (Definidas en el Anexo No.2) es posible generalizar y construir la familia de m-funciones $\mathbf{PE}(\mathbf{S}, \alpha)$, $\mathbf{PE_2}(\mathbf{S}, \alpha_1, \alpha_2)$, $\mathbf{PE_3}(\mathbf{S}, \alpha_1, \alpha_2, \alpha_3)$, ..., $\mathbf{PE_n}(\mathbf{S}, \alpha_1, \alpha_2, ..., \alpha_n)$, las cuales imprimen al final los símbolos α; α_1, α_2; α_1, α_2, α_3; ...; α_1, α_2, ..., α_n; respectivamente. $\rightarrow$ **S**.

5.2.4.1.2 Descripción

m-config	symbol	behavior	final m-config
$\mathbf{PE_n}(\mathbf{S}, \alpha_1, \alpha_2, ..., \alpha_n)$			$\mathbf{PE}(\mathbf{PE}((... \mathbf{PE}(\mathbf{PE}(\mathbf{S}, \alpha_n), \alpha_{n-1}), ...), \alpha_2), \alpha_1)$

5.2.4.1.3 Observaciones

Técnicamente no es correcto afirmar que $\mathbf{PE_n}(\mathbf{S}, \alpha_1, \alpha_2, ..., \alpha_n)$ es una m-función. En realidad es una meta-m-función que permite representar cualquier m-función de la forma $\mathbf{PE_n}(\mathbf{S}, \alpha_1, \alpha_2, ..., \alpha_n)$. Esta meta-m-función se convierte en m-función en el momento en que es invocada, es decir, en el momento en que se conoce el valor de n.

[10] Esta descripción se realiza con base en la metodología descrita y usada en el anexo 2.

5.2.4.2 Descripción-m-función *imprimirInstrucción(**S**,* instrucción)[11]

5.2.4.2.1 Explicación

La m-función **imprimirInstrucción**(**S,** *instrucción*) imprime los símbolos de la cadena *instrucción*. →**S**.

5.2.4.2.2 Descripción

M-CONFIGURACIÓN	SÍMBOLO	COMPORT.	M-CONFIGURACIÓN FINAL
imprimirInstrucción(**S**, *instrucción*)			**PE$_n$**(**S**, *instrucción*)

5.2.4.2.3 Observaciones

Para la descripción de esta m-función se ha supuesto que la cadena de símbolos *instrucciones* está compuesta por los símbolos $\alpha_1\alpha_2 \ldots \alpha_n$.

5.2.4.3 Descripción m-función *marcarInstrucción(**S**, α)*

5.2.4.3.1 Explicación

La m-función **marcarInstrucción**(**S**, α) marca con el símbolo α la instrucción que está a su derecha. → **S**.

5.2.4.3.2 Descripción

M-CONFIGURACIÓN	SÍMBOLO	COMPORTAMIENTO	M-CONFIGURACIÓN FINAL
marcarInstrucción(**S**, α)	D	R, Pα, R	**marca$_2$**
marca$_2$	A	R, Pα, R	**marca$_2$**
	D	R, Pα, R	**marca$_3$**
marca$_3$	C	R, Pα, R	**marca$_3$**
	D	R, Pα, R	**marca$_4$**
marca$_4$	C	R, Pα, R	**marca$_4$**
	L	R, Pα, R	**marca$_5$**
	R	R, Pα, R	**marca$_5$**
	N	R, Pα, R	**marca$_5$**
marca$_5$	D	R, Pα, R	**marca$_6$**
marca$_6$	A	R, Pα, R	**marca$_6$**
	No A	R, R	**S**

[11]El parámetro '**S**' representa el estado al cual se retorna una vez finalizada la m–función.

6 ¿POR QUÉ ES IMPOSIBLE CONSTRUIR UNA MÁQUINA DE TURING DINÁMICA?

Se espera que el título de este capítulo sorprenda al lector. Esta sorpresa está sustentada en el desarrollo realizado hasta el momento; en particular los resultados ofrecidos por el capítulo precedente, que ilustraban la simplicidad de codificar las m-funciones usadas por la m-función **cambiarInstrucción**, la cual como se mencionó, es el núcleo de la MTD.

Se recuerda al lector que en la introducción se indicó que por una parte se deseaba presentar los resultados obtenidos, pero también se deseaba ilustrar la génesis de los mismo, considerando el proceso como los resultados, aspectos fundamentales. Esta tesis hubiera sido mucho más corta si simplemente se hubiera mencionado que es una MTD y por qué no es posible construirla; pero el convencimiento que los resultados de posibilidad como de imposibilidad son igualmente valiosos y estos últimos pierden gran parte de su valor si el proceso de génesis no es ilustrado, motivó la presentación realizada.

La razón de no constructibilidad de la MTD es que la MTD y la MUT manejan **por definición** diferentes códigos para representar los mismos símbolos. La MTD debe desplazarse por el área de la MUT para modificar los símbolos de su descripción, pero estos símbolos están codificados en un código diferente al utilizado por la MTD, luego la MTD es incapaz de modificar su descripción. Se puede afirmar que es una incapacidad sintáctica, la MTD no es capaz de hablar el lenguaje en que ella es codificada para ser ejecutada por la MUT.

A manera de ilustración:

Suponga que la MTD debe cambiar el símbolo ‘ ‘ por el símbolo ‘C’, este cambio podría presentar si de desea modificar el símbolo-escribir ‘0’ por el símbolo-escribir ‘1’, lo cual correspondería a modificar en la descripción de la MTD la cadena de símbolos ‘D C’ por la cadena de símbolos ‘D C C’ (esto ejemplo se ilustró en la sección 5.2.3: Descripción del comportamiento de la m-función **cambiarInstrucción**).

Sea la siguiente codificación para los símbolos de la MTD:

	D
0	D C
1	D C C
A	D C C C
C	D C C C C
L	D C C C C C
N	D C C C C C C
R	D C C C C C C C
;	D C C C C C C C C

La instrucción que cambia el símbolo ‘ ’ por el símbolo ‘C’ en la MTD tendría la forma:
“q_x C N q_{x+1}”

Para efectos de la ilustración supongamos que la instrucción es:
“q_1 C N q_2”.

La codificación de esta instrucción para ser ejecutada por la MUT es:
“D A D D C C C C N D A A”

Pero esta codificación **representa** los símbolos que se desean cambiar ‘ ’ y ‘C’ por las secuencias de símbolos ‘D’ y ‘D C C C C’ respectivamente. Es decir, mientras que la MTD opera con el símbolo ‘C’ la MUT opera con la secuencia ‘D C C C C’. Por esto se afirma que la MTD es incapaz de acceder a los símbolos ‘A’, ‘C’, ‘D’, ‘L’, ‘N’, ‘R’, ‘;’ que conforman su descripción, porque este acceso es a través de la MUT la cual los transforma en las secuencias ‘D C C C’,’D C C C C’, ‘D C C C C C’, ‘D C C C C C C’, ‘D C C C C C C C’, ‘D C C C C C C C C’ y ‘D C C C C C C C C C’, respectivamente.

7 CONCLUSIONES

Aunque se obtuvo un resultado de imposibilidad en la construcción de la MTD, las consecuencias que se desprenden de esta experiencia, presentan un alto grado de fertilidad.

Inicialmente se menciona que las reflexiones elaboradas a partir de la posibilidad de la construcción de la MTD son independientes de si es o no es posible construirla. Lo cual significa que ellas no pierden su valor por el resultado obtenido, por el contrario, las posiciones del autor con respecto al aceptar la complejidad y la incompletitud de una nueva noción de computabilidad se mantienen y orientaran su búsqueda de una nueva noción.

A partir de las lecturas realizadas, se observó que aunque la noción de computabilidad tiene un *status* muy bien definido en la actualidad, diferentes áreas de la ciencia "arremeten" contra ella. Los resultados obtenidos en la elaboración de sistemas biológicos, en la elaboración de máquinas cuánticas y la vigencia de conceptos como las máquinas no triviales, la autorreferencia, la recursividad; permiten prever la emergencia de una nueva noción de computabilidad.

El resultado de imposibilidad obtenido observado desde lo lejos es bastante *trivial*, pero esta trivialidad es la trivialidad de los problemas resueltos, cualquier problema resuelto pierde su *status* de problema y pasa a convertirse en algo trivial (para quien lo resuelve) y es común olvidar el problema que era el problema antes de encontrar su solución *trivial*.

Se espera que la presentación de los resultados obtenidos en relación con la construcción de la MTD ahorre tiempo al lector que tenga una idea semejante, pero principalmente ofrezcan un punto de partida, ya sea para ser refutados, corregidos o ampliados por un lector interesado en el tema. Realmente sería muy grato el conocer que en los resultados presentados existía un error que fue corregido por otro y le permitió ampliar la definición de computabilidad, o que estos resultados gatillaron una nueva idea de como podría ser esta ampliación.

En un plano más general, se destaca el resultado de imposibilidad obtenido. El paradigma de la productividad ha alcanzado algunos de los espacios académicos de la sociedad y preguntas tales como: ¿para qué sirve?, ¿qué hace?, ¿si será posible?, etc.; se anteponen a proyectos que no tienen porque conocer *a-priori* las respuestas a estas preguntas. De haber conocido de antemano que la MTD no se podía construir, esta tesis no habría tenido sentido, pero este es realmente el punto importante: antes no se conocía sí se podía o no se podía construir una MTD, ahora sí se conoce que no es posible construirla y se conoce el porque de esta imposibilidad. Desde una punto de vista general, este resultado representa un muy pequeño trayecto recorrido en el largo camino de las ideas, reconociendo que este camino no es lineal ni tiene una orientación definida. Los trayectos construidos hacia adelante, hacia atrás, en círculos, hacia arriba o hacia abajo, son importantes en cuanto a lo que son y no por la dirección que siguen.

8 BIBLIOGRAFÍA

[Bobenrieth, 1996] BOBENRIETH, M. Andrés. ***Inconsistencias ¿por qué no?.*** 1ed. Santafé de Bogotá, Colombia: Tercer Mundo Editores, División Gráfica. 1996. págs. xxxviii + 567. (Premio Nacional de Colcultura (Instituto Colombiano de Cultura).

[Caicedo, 1990] CAICEDO, Xavier. ***Elementos de lógica y calculabilidad***. 2ed. Santafé de Bogotá: Una empresa docente, Universidad de los Andes. 1990. págs. 254 - 321.

[Delahaye, 1996] DELAHAYE, Paul. ***Creaciones informáticas: Un juego universal de herramientas de cálculo***. En: Investigación y Ciencia, abril 1996. pág. 80 – 83.

[Deutsch-1985] DEUTSCH, D. ***Quantum theory, the Church-Turing principle and the universal quantum computer***. En: Proc. R. Soc. London. A 400. pág. 97 – 117.

[Gómez, 1997] GÓMEZ, Raúl. ***Meditaciones: De los mathématas lógico-textuales o de los juegos de la complejidad***. Ponencia presentada en el simposio: Complejidad: Trabajo transdisciplinario para mejorar la producción y la creatividad – Edgar Morin. Universidad Pontificia Bolivariana. Medellín. Febrero 26, 27 y 28.

[Eco, 1977] ECO, Umberto. ***Cómo se hace una tesis***. Barcelona: Gedisa Editorial. Colección: Libertad y Cambio; Serie Práctica. Traducción: Lucía Baranda y Alberto Clavería Ibañez. pags. 267

[LAROUSSE, 1995] **El pequeño LAROUSSE ilustrado 1996**. Diccionario enciclopédico. 1ed, Segunda reimpresión, Larousse, S.A. 1995. pág. 1792.

[Hamkins, Lewis-1997] HAMKINS, Joel David y LEWIS, Andy. ***Infinite Time Turing Machines***. (El profesor HAMKINS nos envío su artículo por correo electrónico)

[Hermes, 1969] HERMES, Hans. ***Enumerability – Decidability – Computability***. 2ed. New York: Springer-Verlag. págs. 245

[Hopcroft, 1984] HOPCROFT, John. ***Máquinas de Turing***. En: Investigación y Ciencia. Julio 1984. págs. 8-19.

[Kampis, 1992] KAMPIS, George. ***Life-Like Computing Beyond the Machine Metaphor***. En: http://hps.elte.hu/Papers/. págs. 23.

[Kleene, 1974] KLEENE, Stephen C. ***Introducción a la metamatemática***. Madrid: Editorial Tecnos. Colección Estructura y Función. 1974. págs. 495.

[Kuhn, 1962] KUHN, Thomas S. ***La estructura de las revoluciones científicas***. 1ed. en español. 3ª reimpresión. Santafé de Bogotá: Fondo de Cultura Económica. 1996. Colección: Breviarios, No. 213. págs. 320.

[Maturana, Varela, 1996] MATURANA, Humberto y VARELA, Francisco. ***El árbol del conocimiento***. Madrid: Editorial: Debate, S.A. Colección: Pensamiento. pág. 215

[Minsky, 1967] MINSKY, Marvin. ***Computation: Finite and infinite machines***. Englewood Cliffs, N.J.: Prentice-Hall, Inc. págs. 317

[Morin, 1990] MORIN, Edgar. ***Introducción al pensamiento complejo***. 1ed. 2ª reimpresión. Barcelona: Editorial GEDISA. Colección: Ciencias Cognitivas. Traducción: Marcelo Pakman. 1996. pág. 167

[Pattis-1985] PATTIS, Richard. ***El Robot Karel***. México: Editorial Limusa. 1era reimpresión, Traducción: Andrés Eduardo Chehade Duran, 1987. pág. 154.

[Penrose, 1989] PENROSE, Roger. ***La nueva mente del emperador***. Barcelona: Grijalbo Mondadori. Colección: Libro de Mano No. 38. Traducción: Javier García Saenz, 1995. pág. 597

[Penrose, 1994] PENROSE, Roger. ***Las sombras de la mente***. Barcelona: Grijalbo Mondadori. Colección: Drakontos. Traducción: Javier García Saenz, 1996. pág. 480

[Rocha, 1994] ROCHA, Luis Mateus. ***Artificial semantically closed objects***. En: http://ssie.binghamton.edu/~rocha/tilsccai.html. págs. XXX

[Rogers, 1992] ROGERS, Hartley. ***Theory of recursive functions and effective computability***. 3era impresión. Cambridge: The MIT Press. pág. ix + 482.

[Sáez-Fernández, 1987] SÁEZ Fernando y FERNÁNDEZ, Gregorio. ***Fundamentos de informática***. 1ed. Madrid: Alianza Editorial. pág. 550.

[Sicard, 1996] SICARD, Andrés. ***Máquinas de Turing***. En: Revista Universidad EAFIT. No. 103. pág. 29 - 45.

[Sicard, 1997] SICARD, Andrés. ***Máquina universal de Turing: Algunas indicaciones para su construcción***. En: Revista Universidad EAFIT. No. 108. pág. 61 - 106.

[Sieg, 1997] SIEG, Wilfred. ***Step by recursive step: Church's analysis of effective calculability***. En: The bulletin of Symbolic Logic, Volume 3, Number 2, June 1997. pág. 154 - 180.

[Soare, 1996] SOARE, Robert I. **Computability and Recursion**. En: The Bullerin of Symbolic Logic. Volume 2, Number 3, Sept. pág. 284 – 321

[Thuillier, 1988] THUILLIER, Pierre***. De Arquímedes a Einstein: Las caras ocultas de la invención científica***. Madrid: Alianza Editorial, S.A. Sección: Ciencia y Técnica. Tomo I, No. 1487 y Tomo II, No. 1488. Traducción: Amalia Correa, 1990.

[Turing, 1936-37] TURING, Alan. ***On computable numbers, with an application to the Entscheidungsproblem***. En: Proc. London Math. Soc. ser. 2, vol. 42. 1936-1937. págs. 230 - 265. A correction, ibid, vol 43. 1936-1937. págs. 544 - 546.

[von Bertalanffy-1968] von BERTALANFFY, Ludwing. ***Teoría general de los sistemas.*** Colombia: Fondo de Cultura Económica (FCE). 1era reimpresión. Traducción: Juan Almela. 1994

[von Foerster-1984] von FOERSTER, Heinz. ***Principios de autoorganización en un contexto socioadministrativo***. En: Las semillas de la cibernética. Barcelona: Gedisa. Colección: Terapia Familiar. Editor y Traductor: Marcelo Pakman. 1996

[von Foerster-1990] von FOERSTER, Heinz. ***Lethology: A theory of learnig and knowing vis a vis Undeterminables, Undecidables, Unknowables.*** En: Revista Universidad EAFIT. No. 107; julio, agosto, septiembre 1997. pág. 15 - 32.