¿ Máquina de Turing Autorreferencial: Una Imposibilidad ?

Andrés Sicard Ramírez
Departamento de Ciencias Básicas
Universidad EAFIT
Medellín, Colombia, S.A.
email: asicard@eafit.edu.co

Resumen

La sección 1 construye el marco teórico necesario para que la noción de Máquina de Turing Autorreferencial (MTAR) se instale en un contexto científico-filosófico-teórico. La sección 2 define que es una Máquina de Turing (MT), explica un elemento auxiliar para la construcción de las mismas, llamado m-funciones, indica un sistema de codificación y enumeración para las máquinas de Turing y presenta la Máquina Universal de Turing (MUT). La sección 3 describe que es una MTAR. La sección 4 se pregunta por las consecuencias de este constructo teórico; en particular, se pregunta por la relación que existe entre una MT y una MTAR, relación observada desde un único punto de vista: la relación de potencia entre las mismas. La sección 5 se presenta los elementos necesarios para construir una MTAR; para algunos casos, se construyen éstos a manera de "prueba" de la viabilidad de la construcción de la MTAR. La sección 6 ilustra la imposibilidad de construir la MTAR y se convierte en el núcleo de este artículo. Finalmente, la sección 7 presenta algunas conclusiones con base en los resultados obtenidos y se sugieren nuevas posibilidades de abordar el problema.

1. Marco Teórico

A partir de la definición (informal) de computabilidad ofrecida por Soare: *"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, ..."* ([8], pág. 286)

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-. La clase de objetos que permite la pregunta por la computabilidad o no de sus miembros es muy heterogénea; 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 el artículo fundacional de las máquinas de Turing [9]) y en una instancia de mayor cobertura, de procesos computables o no computables (objeto seleccionado por Penrose y tratado exhaustivamente en [3] y [4]). Una particularidad muy significativa de la propiedad de computabilidad, es su carácter binario excluyente, es decir, un objeto es o no es computable, esta característica genera una partición del universo de los objetos.

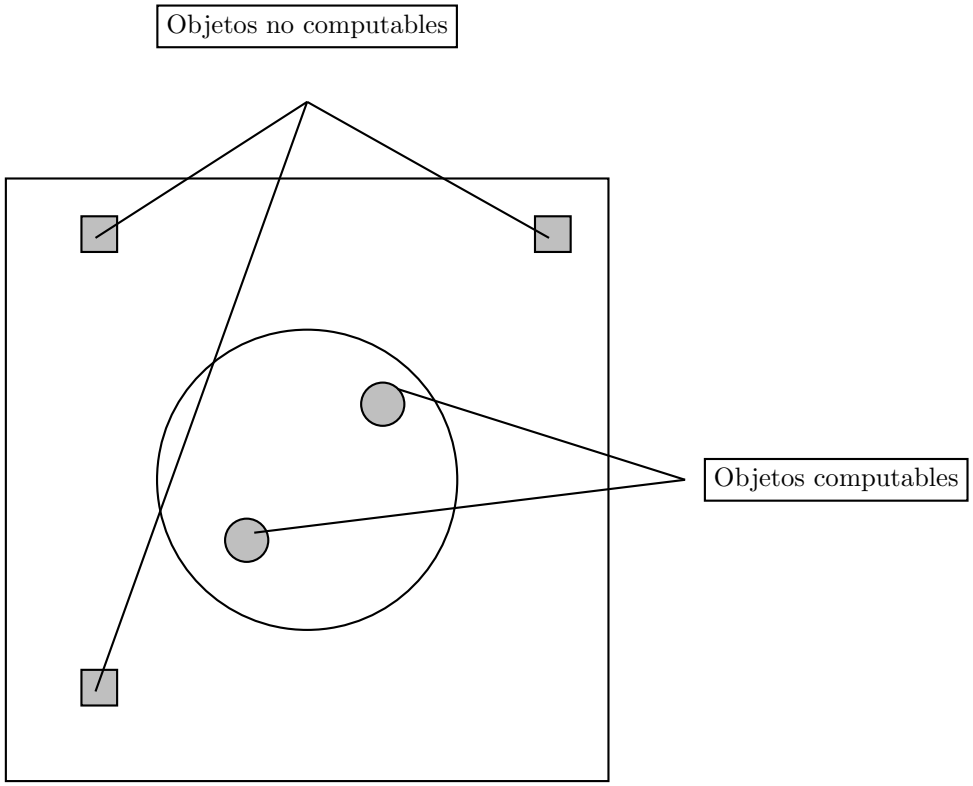


Figura 1: Partición del universo de los objetos: Objetos computables y objetos no computables.

¿ 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 definición misma de computabilidad, pero no una definición informal (como la ofrecida por Soare), sino una definición formal que permita distinguir sin equívoco un objeto computable de un objeto no computable. En la actualidad existen diferentes "versiones" para la definición formal de computabilidad -versiones que como es conocido, satisfacen la propiedad de ser coextensivas-; la versión con la cual se va a operar en este artículo es la correspondiente a las máquinas de Turing: un objeto es computable si es computable por una máquina de Turing, es decir, un objeto es computable si es Turing-computable.

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 retribuir por este "dominio" del objeto es muy

elevado, un objeto computable es un objeto simple, es un objeto trivial (en el sentido de las máquinas triviales de von Foerster [11]).

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 ([9]) y el problema de la teselación ([4]); por el lado factual, se mencionan procesos tales como la morfogénesis ([10]) y la cognición ([4]).

La clasificación binaria ejercida por la noción de computabilidad actúa como un paradigma-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 éstos crucen el filtro y se conviertan así, en objetos aprehensibles; esta es la versión del paradigma de la simplicidad ([2]) observado desde la perspectiva de la computabilidad.

La definición formal de computabilidad ha sido puesta en entredicho desde sus primeras formulaciones. Con alguna frecuencia se han presentado personas que consideran 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 presentan apartes de la carta-respuesta enviada por Alonso Church a Jósez 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, quien es el creador de una de las "versiones" formales de computabilidad (λ-cálculo). Por otra parte, la excelente explicación de Church de las consecuencias de contar con una definición más potente de computabilidad permite justificar el continuar con su búsqueda, aunque también ofrece argumentos bastante sólidos de la imposibilidad de encontrarla; pero de mayor importancia, es la posición de escepticismo adoptada por Church, muy diferente a la posición dogmtica 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.

... 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 extensions. Moreover this new principle of logic must be of so strange, and presumably complicated, a kind that its metamat-

hematical 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).” ([7], pág. 175 - 176)[1]

Este es el contexto en el cual se reflexiona sobre la posibilidad de ampliar la definición de computabilidad: Se presenta inicialmente una representación gráfica (figura 2) de las características que se considera debe poseer una nueva definición de computabilidad.

[1] Algún lector podrá objetar que la propuesta de Pepis no es ampliar la nocin 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.

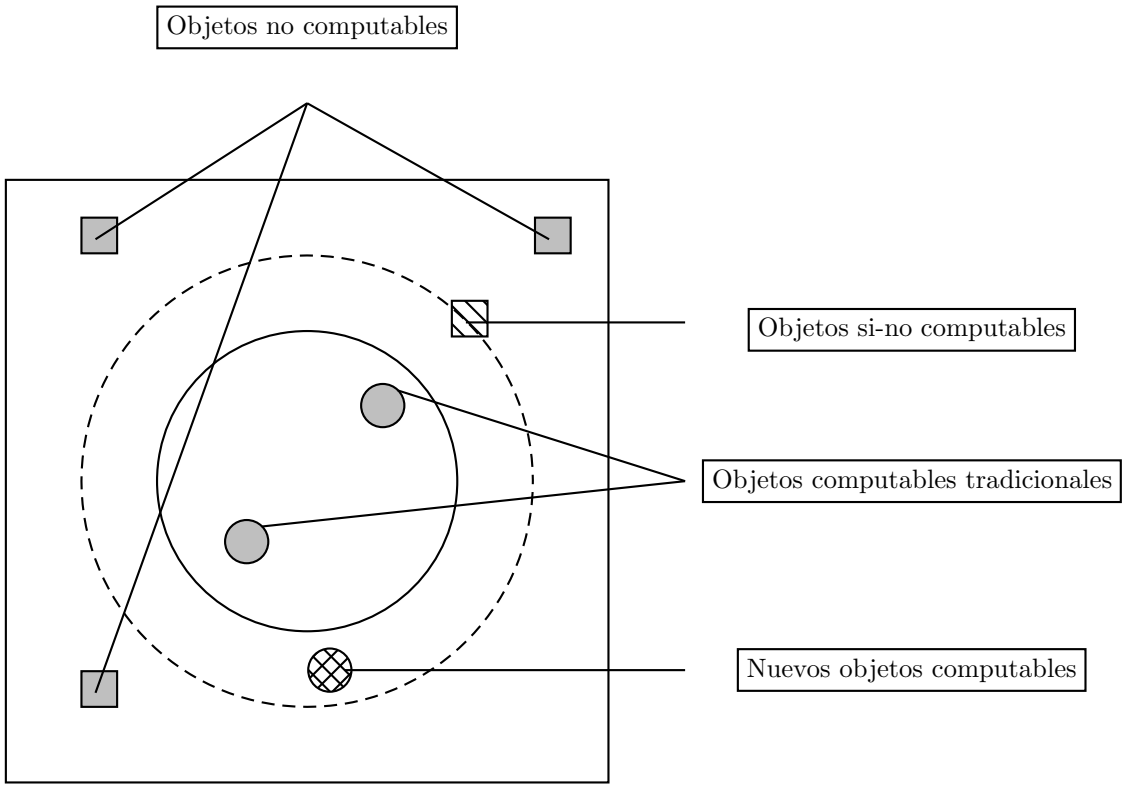


Figura 2: Ampliación de la definición de computabilidad.

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 sílo son (representados por una circulo con rayas). La transformación 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 comparten el mismo grado de no computabilidad, una vez se logre transformar uno de ellos en computable, se lograría transformarlos a todos.[2]

[2]Una vez elaborada esta idea, el autor supo de la existencia de estos grados de no compu-

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 complejos 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, quiebra los principios de la lógica clásica para permitir que se incorporen nuevas posibilidades de verdad. En el contexto de la nueva computabilidad se acepta y se espera la nueva categoría de objetos sí-no computables (representados por cuadrado con rayas). Por supuesto, esto es inadmisible bajo la mirada consistente y binaria de la lógica-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."* (las negrillas son nuestras) ([2], 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 para construir modelos de los sistemas vivos ([5], [1]). Kampis por una parte, a partir del hecho de que la evolución es uno de las principales características de los sistemas vivos y que ésta produce innovaciones en el sistema, las cuales aumentan la complejidad del mismo; y por otra parte, 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 Newto-

tabilidad a partir de [6], bibliografía que una vez revisada, permitirá complementar esta idea.

nian 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 ... and in Turing's Thesis."*** ([8], 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 problemtico 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 definición, 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, surgen 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 uno de sus elementos de 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 lo real, en el cual no es posible triunfar. He aquíel sentido trágico de esta aventura llamada ciencia, pero él mismo constituye su esencia.

2. Algunos elementos acerca de las máquinas de Turing

2.1. Descripción formal de la máquina de Turing

Definimos formalmente una *máquina de Turing deteminística*[3] por una cuadrúpla $\mathfrak{MT} =< Q, \Sigma, M, \delta >$ [4] donde:

$Q = \{q_0, q_1, q_2, \ldots, q_n\}$: Conjunto finito de estados de la máquina ($Q \neq \emptyset$).

$\Sigma = \{s_0, s_1, s_2, \ldots, sm\}$: Alfabeto. Conjunto finito de símbolos de entrada - salida. Adoptamos por convención que $s_0 = \square$ (símbolo vacío) ($\Sigma - s_0 \neq \emptyset$).

$M = \{L, R, N\}$: Conjunto de movimientos (L: izquierda, R: derecha, N: no movimiento).[5]

δ: Es una función definida de un subconjunto $K \times \Sigma$ en $\Sigma \times M \times K$ [6]. δ también puede ser definida como un conjunto *finito* [7] de instrucciones $\delta = \{i_0, i_1, i_2, \ldots, i_p\}$ donde cada i_j es una quíntupla de la forma: $q_m\ s_m\ s_n\ m\ q_n$, donde $q_m, q_n \in K$; $s_m, s_n \in \Sigma$; $m \in M$.

Con respecto a la información inicial que contiene la cinta, la posición inicial de la máquina sobre ésta y el estado inicial de la máquina, se suministran "informalmente" antes de comenzar la ejecución de la máquina.

Si la máquina se encuentra en la situación actual (q_m, s_m) y encuentra una intrucción $i_j : q_m, s_m, s_n, m, q_n$, entonces cambia s_m por s_n, realiza el movimiento indicado por m y pasa al estado q_n (puede ocurrir que $q_m = q_n$ o $s_m = s_n$); de lo contrario la máquina finaliza su ejecución.

[3] Este artículo hace referencia exclusivamente a máquinas de Turing determinísticas, por la tanto la palabra "determinísticas" se omitirá desde este momento.

[4] Esta definición puede tener varias variantes de acuerdo al "gusto" de cada autor. Una de ellas es definir la máquina de Turing como un conjunto finito de instrucciones, ya que este conjunto da implícitamente el conjunto de símbolos utilizados y los posibles estados de la máquina.

[5] Todas las máquinas de este artículo operan sobre un cinta unidimensional bi-infinita. De ahíque este conjunto es el mismo para todas. Hemos utilizado las iniciales de los nombres en ingles de cada movimiento, para poder ilustrar la codificación y la enumeración originales construidas por Turing para las máquinas, la cual será presentada en una sección posterior.

[6] El lector debe notar que la función δ está definida sobre un subconjunto de $K \times \Sigma$ porque no es necesario que exista una instrucción para cada una de las situaciones (estado, símbolo) teórica (el conjunto formado por $K \times \Sigma$) en las que puede estar la máquina. Además, el concepto de función refleja la característica de las máquinas de Turing determinísticas.

[7] La finitud de este conjunto está determinada implícitamente por la finitud del conjunto de estados K y la finitud del alfabeto Σ.

2.2. Enumeración y codificación de las máquinas de Turing

Turing contruyó una enumeración muy particular de sus máquinas. Esta enumeración tiene como base un mecanismo de codificación de las mismas. La codificación de las máquinas de Turing es necesaria para definir un procedimiento que sea capaz de ejecutar cualquier MT (en la sección correspondiente a la máquina universal de Turing indicaremos este procedimiento). Veamos entonces, cuál fue la codificación y la enumeración de máquinas propuesta por Turing [8].

De acuerdo a la definición formal de una máquina de Turing, tenemos el siguiente "esquema" para una instrucción:

$q_i\ s_j\ s_k\ m\ q_l$; donde $i, j, k, l \geq 0$ y $m \in M = \{L, R, N\}$.

Se reemplaza cada instrucción de acuerdo a la siguiente codificación:

q_i: Se reemplaza por una D seguida de A repetida i veces
s_j: Se reemplaza por una D seguida de C repetida j veces

Las instrucciones se reescriben utilizando este código y se separan por un ; . Cuando describimos las instrucciones de nuestra máquina utilizando este sistema de codificación, decimos que la máquina está representada en su *descripción estándar* (*standard description* en el original). Las instrucciones de la descripción estándar estarán formadas con símbolos del alfabeto $\Sigma = \{A, C, D, R, L, N, ;\ \}$.

Asociamos a cada símbolo del alfabeto Σ un número natural por medio de la siguiente función $f : \Sigma \to \mathbb{N}$ donde:[9]

$$f(s) = \begin{cases} 1 & \text{si} \quad s = A \\ 2 & \text{si} \quad s = C \\ 3 & \text{si} \quad s = D \\ 4 & \text{si} \quad s = L \\ 5 & \text{si} \quad s = R \\ 6 & \text{si} \quad s = N \\ 7 & \text{si} \quad s =; \end{cases}$$

Si se reemplaza cada símbolo de la descripción estándar de una máquina de Turing por el valor que tiene asociado el número obtenido es llamado el *número de descripción* (*description number* en el original) de la máquina en cuestión.

[8] En la actualidad se han propuesto otras codificaciones y enumeraciones para las máquinas de Turing.

[9] Esta es la razón por haber escogido el nombre de los movimientos en ingles.

Como ejemplo, hallemos la descripción estándar y el número de descripción para una máquina $\mathfrak{MT}$.

El conjunto de instrucciones de $\mathfrak{MT}$ consta de:

i_1: q_0 □ / R q_1
i_2: q_1 / / R q_1
i_3: q_1 □ □ L q_2
i_4: q_2 / □ N *stop*

Renombremos los símbolos utilizados por $\mathfrak{MT}$ así: □ por s_0 y / por s_1, además renombremos el estado *stop* por el estado q_3. Ahora podemos escribir de nuevo las instrucciones y obtener la descripción estándar de cada una de ellas:

	Instrucción original	Instrucción renombrada	Instrucción codificada
i_1	q_0 □ / R q_1	q_0 s_0 s_1 R q_1	$DDDCRDA$
i_2	q_1 / / R q_1	q_1 s_1 s_1 R q_1	$DADCDCRDA$
i_3	q_1 □ □ L q_2	q_1 s_0 s_0 L q_2	$DADDLDAA$
i_4	q_2 / □ N *stop*	q_2 s_1 s_0 N q_3	$DAADCDNDAAA$

Con lo cual la descripción estándar para la máquina $\mathfrak{MT}$ es:

$DDCDCRD$; $DDDCRDA$; $DDDCRDA$; $DADDLDAA$; $DAADCDNDAAA$;

y su número de descripción es:

33232537333253173332531731334311731132363111 7

Podemos observar que para cada máquina de Turing existe un número natural que la representa, más no todo número natural representa una máquina de Turing. Además de acuerdo a la enumeración anterior es posible demostrar que el conjunto de las máquinas de Turing es enumerable (y en particular infinito), tal como lo es el conjunto de los *algoritmos* que se pueden construir sobre algún lenguaje (y esto debe ser así, si somos consistentes con la idea que cualquier algoritmo puede ser representado por una máquina de Turing).

2.3. m-funciones

2.3.1. Introducción

En la actualidad, el comportamiento de cualquier máquina de Turing es expresado en la "notación de instrucciones". Cada instrucción está constituida por una quíntupla: q_i s_j s_k m q_l donde:

q_i: estado actual

s_j: símbolo que se lee
s_k: símbolo que se escribe
m: movimiento que se realiza
q_l: estado final

Aunque la notación anterior fue propuesta por Turing en su artículo fundacional sobre las máquinas de Turing ([9]), las maquinas construídas por él en dicho artículo (en particular la máquina universal de Turing), no están descritas en la "notación de instrucciones"; por el contrario, están descritas en una notación que permite simplificar considerablemente el número de instrucciones necesarias para describir el comportamiento de una máquina; dicha notación tiene como elemento principal el concepto de **m-función**. Una m-función, se puede interpretar como una máquina de Turing que permite el uso de parámetros. Se presentan a continuación algunos elementos necesarios para poder formalizar (parcialmente) y ejemplificar el uso de m-funciones. Además, se indicaran los procedimientos requeridos para expresar en la "notación de instrucciones" cualquier m-función lo cual permitirá al lector paladear la simplicidad-complejidad ofrecida por éstas.

2.3.2. Definiciones auxiliares

Se presenta algunos términos definidos por Turing, con el objetivo de realizar la correlación con los términos usados en la actualidad y permitir leer las explicaciones presentadas por Turing en lo que respecta al hacer de las m-funciones por él creadas, que nos serviran como ejemplo.

m-configuration: *"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* $q_1, q_2, \ldots, q_R$ *which will be called "m-configurations"".* ([9], pág. 231)
El término "m-configuración" es remplazado en la actualidad por el término "estado".

configuration: *"The posible behavior of the machine at any moment is determined by the m-configuration* q_n *and the scanned symbol* $S(\gamma)$*. This pair* $q_n, S(\gamma)$ *will be called the "configuration": thus the configuration determines the possible behavior of the machine."* ([9], pág. 231)
El término "configuración" es reemplazado en la actualidad por el término "situación actual".

F-squares y E-squares: *"The convention of writing the figures only on alternate squares is very useful: I shall always make use of it. I shall call the one sequence of alternate squares F-squares and the other sequence E-squares. The symbols on E-squares will be liable to erasure. The symbols on F-squares form a continuous sequence."* ([9], pág. 235)

Marking a symbol: *"If a symbol* β *is on an F-square S and a symbol* α *is*

on the E-square next on the right of S, then S and β will be said to marked with α. The process of printing this α will be called marking β (or S) with α." ([9], pág. 235)

2.3.3. Definición m-función

Se presenta la definición y el ejemplo ofrecido por Turing, en relación con este concepto.

There are certain type of process used by nearly all machines, and these, in some machines, are used in many connections. These processes include copying down sequence of symbols, comparing sequences, erasing all symbols of a given form, etc. Where such processes are concerned we can abbreviate the tables for the m-configurations considerably by the use of "skeleton tables". In skeleton tables there appear capital German letters [10] *and small Greek letters. These are of the nature "variables". By replacing each capital German letter throughout by an m-configuration and each small Greek letter by a symbol, we obtain the table for an m-configuration."*

Let us consider an example:

m-config	**symbol**	**behavior**	**final m-config**
$\boldsymbol{F}(\boldsymbol{S},\boldsymbol{B},\alpha)$	e	L	$\boldsymbol{F_1}(\boldsymbol{S},\boldsymbol{B},\alpha)$
	not e	L	$\boldsymbol{F}(\boldsymbol{S},\boldsymbol{B},\alpha)$
$\boldsymbol{F_1}(\boldsymbol{S},\boldsymbol{B},\alpha)$	α		$\boldsymbol{S}$
	not α	R	$\boldsymbol{F_1}(\boldsymbol{S},\boldsymbol{B},\alpha)$
	$None$	R	$\boldsymbol{F_2}(\boldsymbol{S},\boldsymbol{B},\alpha)$
$\boldsymbol{F_2}(\boldsymbol{S},\boldsymbol{B},\alpha)$	α		$\boldsymbol{S}$
	not α	R	$\boldsymbol{F_1}(\boldsymbol{S},\boldsymbol{B},\alpha)$
	$None$	R	$\boldsymbol{B}$

"If we were to replace $\boldsymbol{S}$ throughout by $\boldsymbol{Q}$ (say), $\boldsymbol{B}$ bye $\boldsymbol{R}$, and α by χ, we should have a complete table for the m-configuration $\boldsymbol{F}(\boldsymbol{Q},\boldsymbol{R},\chi)$. $\boldsymbol{F}$ is called an "m-configuration function" or "m-function"." ([9], pág. 235-236).

Como se indicó, el concepto de m-función, se interpreta como una máquina de Turing que permite el uso de parámetros (estados y símbolos) en el momento de su definición, los cuales serán instanciados en el momento de su invocación. El uso y la combinación de estas máquinas de Turing parametrizadas o m-funciones simplifica la descripción de las máquinas que las usan.

[10] Se utilizará letras mayúsculas en negrilla en lugar de las letras máyusculas alemanas tanto para denotar algunas "variables" (las que correspondan a m-configuraciones) de las m-funciónes, como para denotar el nombre de las mismas.

2.3.4. Notación no tradicional

Como se mencionó con anterioridad, la descripción de algunas de las máquinas propuestas por Turing se realiza con base en las m-funciones y en una "notación no tradicional" (diferente a la "notación de instrucciones"). Esta "notación no tradicional" es presentada por medio de los siguientes ejemplos (ofrecidos por Turing):

Es frecuente en Turing el utilizar una notación especial, para escribir o borrar un símbolo, como lo indican los siguientes ejemplos:

m-config	**symbol**	**behavior**	**final m-config**
B	E		B

En este caso, se elimina el símbolo sobre el que está la máquina.

m-config	**symbol**	**behavior**	**final m-config**
B		$P\alpha$	B

En este caso se escribe el símbolo α sobre la cinta.

Turing también utilizaba una notación que permitía en la columna de *operations* varios símbolos, como lo indican los siguientes ejemplos:

m-config	**symbol**	**behavior**	**final m-config**
B	$None$	$P0, L$	B

En la notación actual, se podría escribir:

estado	**símbolo leer**	**símbolo escribir**	**movimiento**	**estado siguiente**
B	$\square$ [11]	0	L	B

Ejemplo:

m-config	**symbol**	**behavior**	**final m-config**
B	0	$R, R, P1$	B

Que en la notación actual sería:

estado	**símbolo leer**	**símbolo escribir**	**movimiento**	**estado siguiente**
q_i	0	0	R	q_{i+1}
q_{i+1}	$cualquiera*$	$cualquiera*$	R	q_{i+2}
q_{i+2}	$cualquiera*$	1	N	q_i

* El ejemplo anterior permite entrever la dificultad que se presenta el traducir la notación utilizada por Turing a la "notación de instrucciones". El doble movimiento indicado por la operación $R, R, P1$ para ser escrito en la "notación

[11] El símbolo $None$ será representado por el símbolo $\square$

tradicional", exige conocer el conjunto de símbolos que pueden estar sobre la cinta; este conjunto de símbolos se denota por el símbolo *cualquiera*.

La "notación no tradicional" permite escribir en una sola instrucción, el número de operaciones que se desee, como lo permite observar el siguiente ejemplo:

m-config	**symbol**	**behavior**	**final m-config**
B		$Pe, R, Pe, R, P0$ $R, R, P0, L, L$	C

En este caso a partir de la m-configuración B, sin importar sobre cual símbolo del alfabeto se encuentre la máquina, ésta imprime el símbolo e, se mueve a la derecha, imprime de nuevo el símbolo e, se mueve a la derecha, imprime el símbolo 0, se mueve dos casillas a la derecha, imprime el símbolo 0, finalmente se mueve dos casillas a la izquierda y pasa a la m-configuración C. Se deja al lector traducir lo anterior a la "nomenclatura de instrucciones".

2.3.5. Ejemplos m-funciones

Se presentan a continuación tres ejemplos de m-funciones propuestas por Turing. Por cada m-función presentada se seguirán los siguientes pasos:

Nombre m-función:
Se indica el nombre de la m-función que se va a "traducir".

Explicación original:
Se presenta la explicación ofrecida por Turing para la m-función en cuestión.

Descripción original:
Se presenta la descripción original ofrecida por Turing para la m-función en cuestión.

Observaciones:
Se indican algunas observaciones para la m-función en cuestión (esta sección será opcional)

Expansión:
La "traducción final" de una m-función a la "nomenclatura de instrucciones" (instrucciones de la forma: $q_i\ s_j\ s_k\ m\ q_l$) exige conocer el valor de los parámetros con los cuales es invocada y el alfabeto de la máquina de la cual la m-función hace parte. Lo que si es posible realizar con base en la definición de una m-función, es una "traducción parcial" a la "nomenclatura de instrucciones" que facilite el proceso de "traducción final" una vez se conozca la información (valor de los parámetros y alfabeto de la máquina) requerida para ello; este proceso de "traducción parcial" es denominado la *expansión* de la m-función.

La expansión de la m-función se presenta bajo el formato:

estado actual	**símbolo leer**	**símbolo escribir**	**mov**	**estado siguiente**	**explicación**

Ejemplo 2.1. ***m-función*** $\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$

Explicación original:
"From the m-configuration $\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$ *the machine finds the symbol of form* α *which is farthest to the left (the "first* α*") and the m-configuration then becomes* $\boldsymbol{S}$*. If there is no* α *then the m-configuration becomes* $\boldsymbol{B}$*." ([9], pág. 236)*

Descripción original

m-config	***symbol***	***behavior***	***final m-config***
$\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$	e	L	$\boldsymbol{F_1}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$
	not e	L	$\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$
$\boldsymbol{F_1}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$	α		$\boldsymbol{S}$
	not α	R	$\boldsymbol{F_1}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$
	$None$	R	$\boldsymbol{F_2}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$
$\boldsymbol{F_2}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$	α		$\boldsymbol{S}$
	not α	R	$\boldsymbol{F_1}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$
	$None$	R	$\boldsymbol{B}$

Observaciones:
Es frecuente que una m-función -en este caso, la m-función $\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$*- esté compuesta por varias m-funciones -para este caso* $\boldsymbol{F_1}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$ *y* $\boldsymbol{F_2}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$*.*

El símbolo 'e' lo utiliza Turing para indicar el comienzo de los datos en la cinta.

Entre las definiciones de las m-funciones $\boldsymbol{F_1}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$ *y* $\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$ *se observa que la definición de* $\boldsymbol{F_1}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$*:*

m-config	***symbol***	***behavior***	***final m-config***
$\boldsymbol{F_1}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$	α		S
	not α	R	$\boldsymbol{F_1}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$
	$None$	R	$\boldsymbol{F_2}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$

Explícita el comportamiento para el símbolo $None$ *(*$\square$*); por el contrario, la definición para* $\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$*:*

m-config	***symbol***	***behavior***	***final m-config***
$\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$	e	L	$\boldsymbol{F_1}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$
	not e	L	$\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$

No lo explícita. En este caso el comportamiento del símbolo $None$ *(*$\square$*) está implicito en el comportamiento para el "símbolo" not e, tal como lo indica la expansión de* $\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$*.*

Expansión de $\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$

estado actual	***símbolo leer***	***símbolo escribir***	***mov***	***estado siguiente***	***explicación***
q_i	e	e	L	q_{i+1}	*Expansión para la m-función* $\boldsymbol{F}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$
q_i	$\neg e$ [12]	$\neg e$	L	q_i	
q_i	$\square$	$\square$	L	q_i	
q_{i+1}	α	α	N	$\boldsymbol{S}$	*Expansión para la m-función* $\boldsymbol{F_1}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$
q_{i+1}	$\neg\alpha$	$\neg\alpha$	R	q_{i+1}	
q_{i+1}	$\square$	$\square$	R	q_{i+2}	
q_{i+2}	α	α	N	$\boldsymbol{S}$	*Expansión para la m-función* $\boldsymbol{F_2}(\boldsymbol{S}, \boldsymbol{B}, \alpha)$
q_{i+2}	$\neg\alpha$	$\neg\alpha$	R	q_{i+1}	
q_{i+2}	$\square$	$\square$	R	$\boldsymbol{B}$	

Ejemplo 2.2. ***m-función*** $\boldsymbol{PE}(\boldsymbol{S}, \beta)$

Explicación original:
"From $\boldsymbol{PE}(\boldsymbol{S}, \beta)$ *the machine prints* β *at the end of the sequence of sysmbols and* $\longrightarrow S$*" ([9], pág. 237)*

Descripción original

m-config	***symbol***	***behavior***	***final m-config***
$\boldsymbol{PE}(\boldsymbol{S}, \beta)$			$\boldsymbol{F}(\boldsymbol{PE_1}(\boldsymbol{S}, \beta), \boldsymbol{S}, e)$
$\boldsymbol{PE_1}(\boldsymbol{S}, \beta)$	*Any*	R, R	$\boldsymbol{PE_1}(\boldsymbol{S}, \beta)$
$\boldsymbol{PE_1}(\boldsymbol{S}, \beta)$	*None*	$P\beta$	$\boldsymbol{S}$

Observaciones:
Para simplicar la expansión de la m-función $\boldsymbol{F}(\boldsymbol{PE_1}(\boldsymbol{S}, \beta), \boldsymbol{S}, e)$ *se supondrá el símbolo* $''e''$ *está en la cinta.*

Expansión de $\boldsymbol{PE}(\boldsymbol{S}, \beta)$

[12] El símbolo $\neg e$ representa el conjunto de símbolos del alfabeto $\Sigma - \{e\}$.

estado actual	***símbolo leer***	***símbolo escribir***	***mov***	***estado siguiente***	***explicación***
q_i	e	e	L	q_{i+1}	*Expansión para la m-función* $\boldsymbol{PE}(\boldsymbol{S}, \beta)$
q_i	$\neg e$	$\neg e$	L	q_i	
q_i	$\square$	$\square$	L	q_i	
q_{i+1}	e	e	N	q_{i+2}	*Invocación para la m-función* $\boldsymbol{PE}(\boldsymbol{S}, \beta)$
q_{i+2}	$Cualquiera$	$Cualquiera$	R	q_{i+3}	*Expansión para la m-función* $\boldsymbol{PE}(\boldsymbol{S}, \beta)$
q_{i+2}	$\square$	β	N	$\boldsymbol{S}$	
q_{i+3}	$Cualquiera$	$Cualquiera$	R	q_{i+2}	
q_{i+3}	$\square$	$\square$	R	q_{i+2}	

Ejemplo 2.3. ***m-función*** $\boldsymbol{PE_2}(\boldsymbol{S}, \alpha, \beta)$

Explicación original:
"$\boldsymbol{PE_2}(\boldsymbol{S}, \alpha, \beta)$*. The machine prints* α β *at the end." ([9], pág. 239)*

Descripción original

m-config	***symbol***	***behavior***	***final m-config***
$\boldsymbol{PE_2}(\boldsymbol{S}, \alpha, \beta)$			$\boldsymbol{PE}(\boldsymbol{PE}(\boldsymbol{S}, \beta), \alpha)$

Expansión de $\boldsymbol{PE_2}(\boldsymbol{S}, \alpha, \beta)$
Se deja al lector el realizar la expansión de esta m-función.

2.4. Máquina universal de Turing

Turing no solo definió formalmente el concepto de algoritmo por medio de sus máquinas de Turing, sino que además (en el mismo artículo donde describió éstas), construyó el modelo teórico de nuestros computadores actuales por medio de una máquina abstracta conocida en nuestros días como *máquina universal de Turing*.

La máquina universal de Turing es una máquina de Turing muy particular. Recibe como entrada una máquina de Turing y una secuencia de datos iniciales. La máquina universal de Turing realiza la ejecución de la máquina de Turing con la secuencia de datos iniciales.

En notación funcional, expresamos esto por: Sean $\mathfrak{U}$ la máquina universal de Turing, $\mathfrak{MT}$ una máquina de Turing, α una secuencia de datos iniciales y β una

secuencia de datos de salida, entonces: $\mathfrak{U}(\mathfrak{MT}, \alpha) = \beta \Longleftrightarrow \mathfrak{MT}(\alpha) = \beta$. En el lenguaje informático lo expresamos por: Un computador (máquina universal de Turing) ejecuta un algoritmo (máquina de Turing) con unos datos de entrada (secuencia de datos iniciales) y obtiene un salida (secuencia de datos de salida).

La máquina universal de Turing es una máquina de Turing (muy particular). Antes de comenzar su ejecución, los datos de entrada (máquina de Turing, secuencia de datos iniciales) deben estar en la cinta. La máquina de Turing actúa como primer dato de entrada, y debe estar expresada en su descripción estándar, a su derecha se escribe la secuencia de datos iniciales. Es decir:

...	Descripción estándar de una $\mathfrak{MT}$		secuencia datos iniciales	...

La máquina universal de Turing está compuesta por un conjunto de m-funciones "saben" cómo "ejecutar" la máquina de Turing (expresada en su descripción estándar) para la secuencia de datos iniciales. La descripción exacta del comportamiento de máquina universal de Turing es bastante compleja y no queremos agobiar al lector con los detalles. Como resumen podemos decir lo siguiente: La máquina universal de Turing debe "capturar" la situación actual (estado actual, símbolo actual) de la máquina de Turing. Entonces debe ir a buscar una instrucción para esta situación actual (en la sección de la descripción estándar de la máquina), si la encuentra debe realizar lo que esta instrucción indica (en la sección de la secuencia de datos iniciales), recordando cual era la posición de la máquina (que está ejecutando) sobre los mismos, entonces de nuevo va y busca una instrucción para la nueva situación actual y asícontinua su proceso, hasta que la máquina de Turing se detenga (si éste es el caso).

Existen varios detalles a tener en cuenta como: ¿ Qué sucede si la máquina de Turing escribe sus resultados a la izquierda de la secuencia de datos iniciales ?. Si éste es el caso, la máquina de Turing despúes de escribir su primer resultado, comenzará a "destruir" su propia descripción estándar, lo cual por supuesto no se debe permitir. Para evitar esta situación se proponen dos soluciones: o se modifica la máquina de Turing para que escriba sus resultados a la derecha de la secuencia de datos iniciales o se modifica la máquina universal de Turing para que suministre tantas celdas en blanco a la izquierda de la secuencia de datos iniciales como necesite la máquina de Turing que está ejecutando.

3. ¿ Qué es una Máquina de Turing Autorreferencial

Una máquina de Turing autorreferencial (MTAR) es una máquina de Turing (MT) en la cual es posible modificar su conjunto de instrucciones. Por supuesto, si se construye una máquina de Turing $\mathfrak{M}$, el diseñador puede modificar algunas instrucciones y construir una nueva máquina de Turing $\mathfrak{M}'$, es decir,

en principio siempre es posible modificar una MT para obtener una nueva MT. Entonces, ¿ qué significa el que en una MTAR sea posible modificar su conjunto de instrucciones?

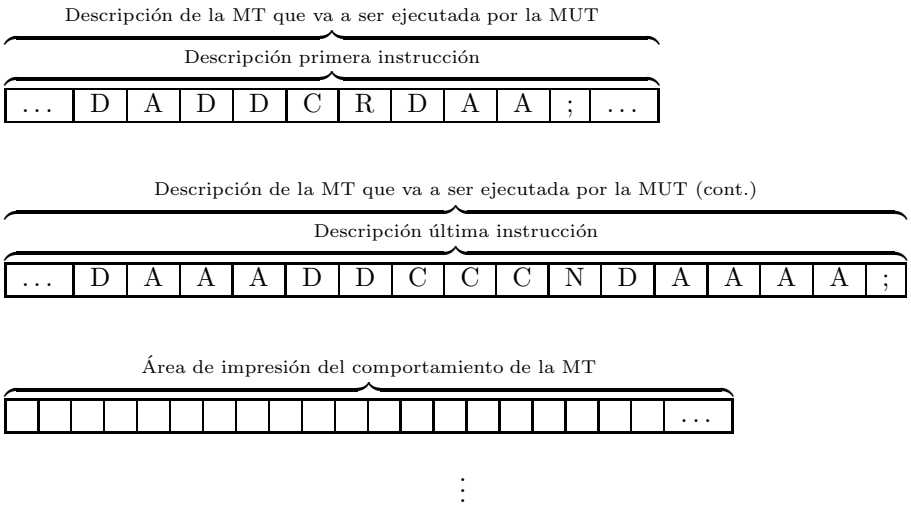
La característica subyacente a la modificación de instrucciones en una MTAR, radica en el agente ejecutor de esta modificación: ella misma. La MTAR está dotada de una propiedad de autorreferencia que le permite modificar su propia descripción-comportamiento.

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 mismos?. Para poder hablar de una MTAR, sería necesario aceptar que los símbolos tomaran lápiz y borrador y escribieran de nuevo sobre el papel, pero 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 MTAR?. Para contestar esta pregunta, es necesario entrar un poco más en detalle en la relación entre la MUT y la MT que ésta ejecuta.

La MUT recibe como parámetro la MT, codificada en el código especial construido por Turing para ello. El comportamiento de una MUT y una MT puede ser esquematizado así:

Inicio ejecución



Final ejecución

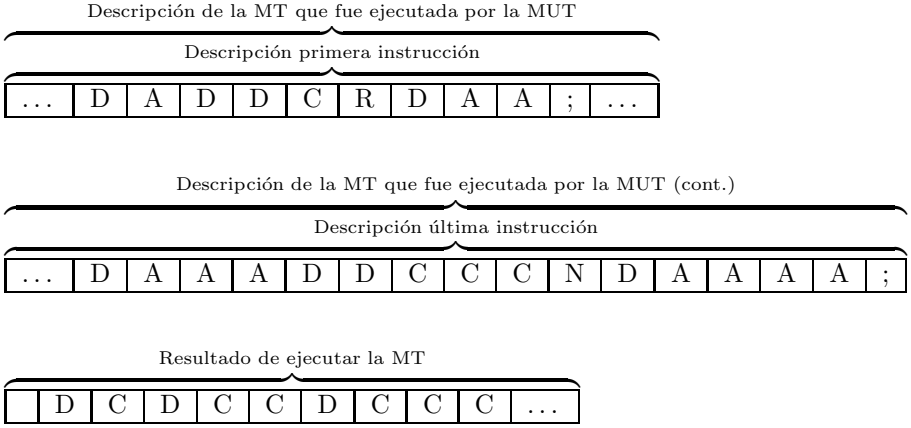


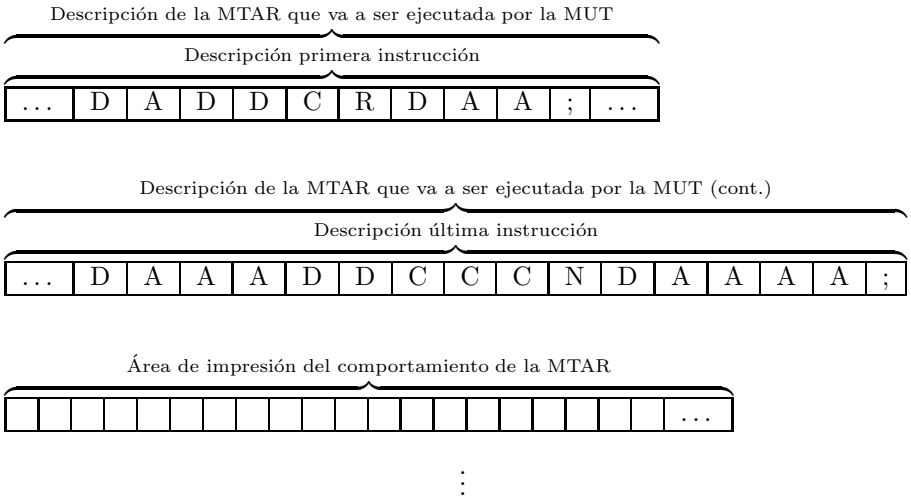
Figura 3: Ejecución de una MT por medio de la MUT.

Para efectos de ilustración, no interesa las instrucciones que componen la máquina $\mathfrak{M}$, ni sus resultados. Lo que síes importante observar es que la descripción de la máquina $\mathfrak{M}$ 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 $\mathfrak{M}$.

Cuando se habla de una MTAR, 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 de ejecución.

El comportamiento de una MUT y una MTAR puede ser esquematizado así:

Inicio ejecución



Final ejecución

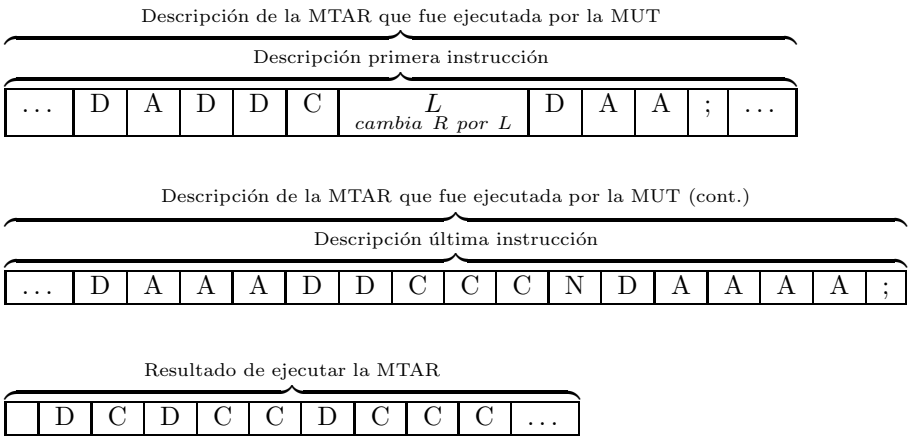


Figura 4: Ejecución de una MTAR por medio de la MUT.

Es decir la máquina MTAR se automodifica y se convierte en una máquina MTAR'. Para poder realizar ésto, la máquina MTAR debe contar con instrucciones especiales que le indiquen como modificarse. De la Figura 4, se puede observar que dichas instrucciones deben poder escribir en la sección correspondiente a la descripción de la máquina MTAR, 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 MTAR es la relación de "potencia". Se entiende por potencia la capacidad del formalismo empleado en calcular objetos, en este sentido se afirma que la máquina de Turing es más potente que una máquina de estado finito, pero que es equipotente con una función recursiva. Entonces, ¿ es la MTAR menos potente, igual de potente o más potente que una MT?. A manera informal se presentan las siguientes consideraciones:

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

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

Finalmente, ¿ es la MTAR más potente que la MT?. El lector debe considerar muy cuidadosamente lo que esto significa, existirían objetos que podrían ser computados por un MTAR, 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 MTAR y la MT en aras de comprender la relación de potencia entre las mismas. Dada la característica de automodificación de la MTAR, está puede ser cualquiera MT en algún momento **discreto** de tiempo. Además, a partir del hecho que para cualquier máquina de Turing existe un número de descripción que la identifica, se puede afirmar que la MTAR 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 y asísucesivamente. La palabra discreto resaltada al comienzo de este párrafo es de suma importancia, si se "fotografía" la ejecución de una MTAR, en cada instante se puede observar una MT diferente [13]. Pero, ¿ que sucede si en lugar de "fotografiar" la MTAR, se "filma" ?, ¿ qué máquina se observa durante la "filmación" ?

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, tal como se indico en la sección 2.1. No sería difícil concebir una MTAR que modifique el estado q_n de alguna instrucción por el

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

estado q_{n+1}, tampoco sería difícil concebir una MTAR 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 MTAR 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 $\aleph_0$ (*aleph-cero*). Si una MTAR puede ser cualquier MT en algún instante, la demostración de mayor potencia de la MTAR sobre la MT se realizaría demostrando un teorema con la siguiente forma:

$$\cup_{i=0}^{\aleph_0} \mathbb{M}_i \succ \mathbb{M}_k \qquad k \text{ constante}$$

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 autorreferencial?

5.1. Introducción

Para la MTAR, 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* (descrita en la sección 2.3). En este contexto los términos instrucción y m-función serán intercalados.

Se espera que una MTAR este formada por un conjunto de m-funciones que representen su comportamiento *tradicional* y por un conjunto de m-funciones que representan su comportamiento *autorreferencial*.

Esta sección está dedicada 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 éstas no "interesan", dado que ellas son "independientes" de las instrucciones autorreferenciales [14].

La característica inicial de las instrucciones autorreferenciales es que estas sean

[14] En realidad síexiste 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.

independientes de las instrucciones de comportamiento. Es decir, se espera poder modificar cualquier instrucción de comportamiento [15].

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. 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 m-función cambiarInstrucción

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 -*instrucciónFina*l-. 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

[15] La posibilidad de autorreferencia de segundo grado, es decir, la posibilidad de modificar las instrucciones de autorreferencia por ellas mismas, no será contemplada.

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 componen 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. 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 modifica 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 modifican el estado-actual, el símbolo-leer y el símbolo-escribir.
"q_i, s_j, s_k, m, q_l"	"$q'_i, s'_j, s'_k, m', q'_l$"	Se modifican todos los elementos de la instrucción-original

5.2.2. Cuerpo m-función cambiarInstrucción

Antes de realizar la descripción de la m-función **cambiarInstrucción** en la metodología y notación empleada para describir las m-funciones, 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.

```
proc cambiarInstruccion(instruccionOriginal, instruccionFinal); ≡
  (1)  guardarPosicionInicial();
  (2)  imprimirInstruccion(instruccionOriginal);
  (3)  marcarInstruccion(o);
  for i := 1 to totalIntrucciones do
     (4)  marcarInstruccion(i);
     (5)  compararInstrucciones(i, o);
     if instruccion_i = instruccionOriginal
       then
            (6)  imprimirInstrucciones(instruccionFinal);
            (7)  marcarInstruccion(f);
            (8)  reemplazarInstruccion(i, f);
            (9)  eliminarInstruccion(f);
            (10)  eliminarInstruccion(o);
            (11)  eliminarMarca(i);
            (12)  restablecerPosicionInicial();
        else (13)  eliminarMarca(i);
     end
  end
  (14)  eliminarInstruccion(o);
  (15)  restablecerPosicionInicial();
end
```

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 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 MTAR la máquina de Turing autorreferencial formada por las siguientes instrucciones:

Instrucciones tradicionales:
$i_1 : q_1 \ \square \ 0 \ N \ q_1$
$i_2 : q_1 \ 0 \ 0 \ R \ q_2$

Instrucciones de autorreferencia:

i_3 : **cambiarInstrucción**($\underbrace{q_1 00 R q_2}_{instruccionOriginal}$, $\underbrace{q_1 01 R q_2}_{instruccionFinal}$)

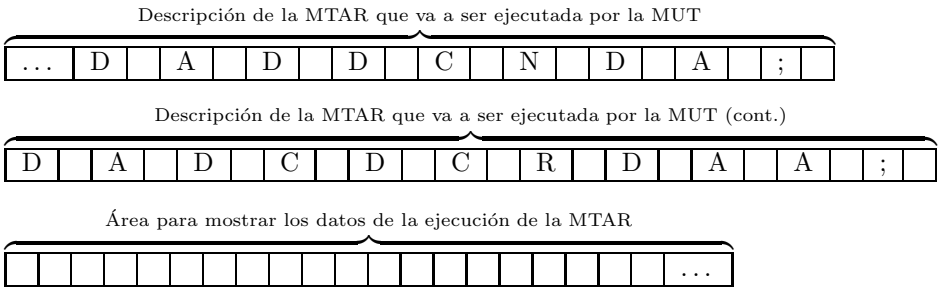
En este caso, en la instrucción i_2 se va a modificar el símbolo-escribir $''0''$ por el símbolo-escribir $''1''$.

La codificación para las instrucciones es:
$i_1 : DADDCNDA$
$i_2 : DADCDCRDAA$

Con lo cual, la m-función **cambiarInstrucción** toma la forma:
cambiarInstrucción($\underbrace{DADDCNDA}_{instruccionOriginal}$, $\underbrace{DADCDCRDAA}_{instruccionFinal}$)

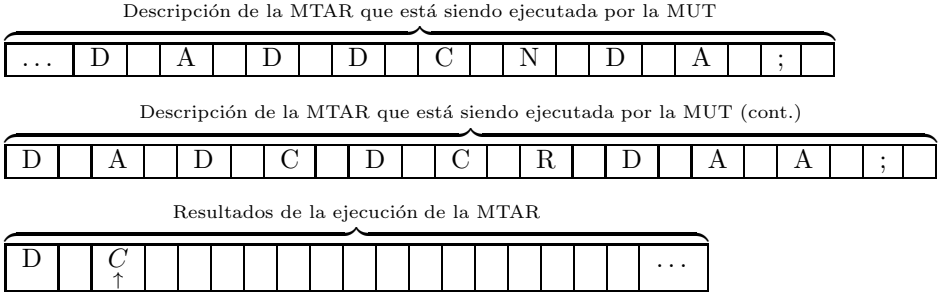
Al inicio de la ejecución de la MTAR por la MUT se tiene:

Inicio



Después de ejecutar las instrucciones i_1 e i_2, se obtiene [16]:

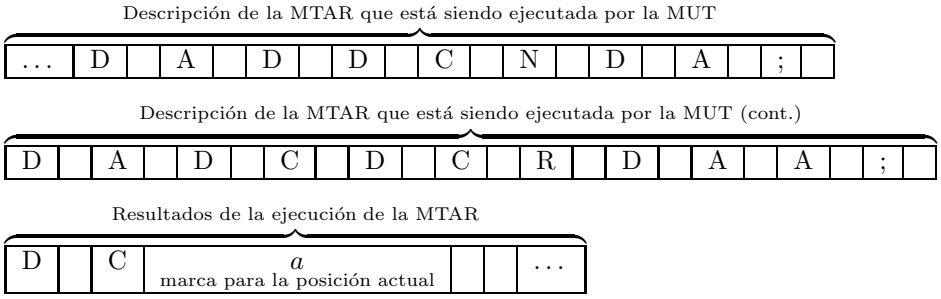
Ejecución instrucciones i_1, i_2



[16] El símbolo ↑ denota la posición actual.

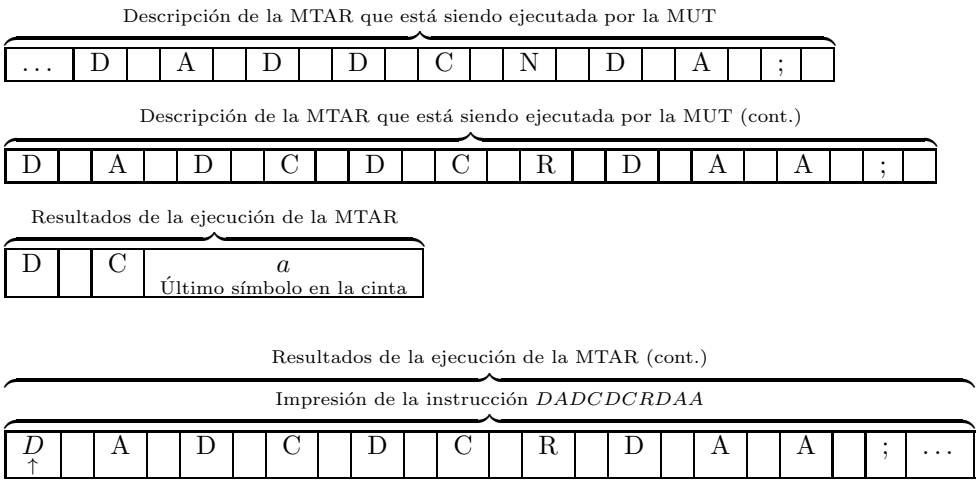
En este punto comienza la ejecución de la m-función **cambiarInstrucción**. Inicialmente se ejecuta la línea (1) **guardarPosiciónInicial()**, en donde la m-función **guardarPosiciónInicial()** marca la posición actual con el símbolo $''a''$, entonces se obtiene:

Ejecución de la m-función guardarPosición()



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 marcarInstrucción(o)

Descripción de la MTAR que está siendo ejecutada por la MUT

...	D		A		D		D		C		N		D		A		;		

Descripción de la MTAR que está siendo ejecutada por la MUT (cont.)

D		A		D		C		D		C		R		D		A		A		;			

Resultados de la ejecución de la MTAR

D		C	a

Resultados de la ejecución de la MTAR (cont.)

Marca de la intrucción $DADCDCRDAA$ con el símbolo o (cont.)

D	o	A	o	D	o	C	o

Resultados de la ejecución de la MTAR (cont.)

Marca de la intrucción $DADCDCRDAA$ con el símbolo o

D	o	C	o	R	o	D	o	A	o	A	o	;	o	...

Las líneas (4) a (13) están dentro de un ciclo que itera una vez por cada instrucción de la MTAR, 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 $''DADDCNDA;\ ''$ 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 $''DADCDCRDAA''$ 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 imprimirInstrucción(D A D C D C C N D A A)

Descripción de la MTAR que está siendo ejecutada por la MUT

...	D		A		D		D		C		N		D		A		;		

Descripción de la MTAR que está siendo ejecutada por la MUT(cont.)

Marca de la intrucción $DADCDCRDAA$ con el símbolo i

D	i A	i	D	i	C	i

Descripción de la MTAR que está siendo ejecutada por la MUT(cont.)

Marca de la intrucción $DADCDCRDAA$ con el símbolo i (cont.)

D	i	C	i	R	i	D	i	A	i	A	i	;	i

Resultados de la ejecución de la MTAR

D		C	a	D	o	A	o	D	o	C	o

Resultados de la ejecución de la MTAR (cont.)

D	o	C	o	R	o	D	o	A	o	A	o	;	o

Resultados de la ejecución de la MTAR (cont.)

Impresión de la intrucción $DADCDCCRDAA$

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:

Ejecución de la m-función marcarInstrucción(f)

Descripción de la MTAR que está siendo ejecutada por la MUT

...	D		A		D		D		C		N		D		A		;	

Descripción de la MTAR que está siendo ejecutada por la MUT(cont.)

D	i A	i	D	i	C	i

Descripción de la MTAR que está siendo ejecutada por la MUT(cont.)

D	i	C	i	R	i	D	i	A	i	A	i	;	i

Resultados de la ejecución de la MTAR

D		C	a	D	o	A	o	D	o	C	o

Resultados de la ejecución de la MTAR (cont.)

D	o	C	o	R	o	D	o	A	o	A	o	;	o

Resultados de la ejecución de la MTAR (cont.)

Marca de la intrucción $DADCDCCRDAA$ con el símbolo f

D	f	A	f	D	f	C	f

Resultados de la ejecución de la MTAR (cont.)

Marca de la intrucción $DADCDCCRDAA$ con el símbolo f (cont.)

D	f	C	f	C	f	R	f	D	f	A	f	A	f	...

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); restablecerPosiciónInicial()

Descripción de la MTAR que está siendo ejecutada por la MUT

...	D		A		D		D		C		N		D		A		;	

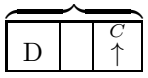
Descripción de la MTAR que está siendo ejecutada por la MUT(cont.)

La intrucción $DADCDCRDAA$ se cambio por la instrucción $DADCDCCRDAA$

D		A		D		C		D		C		C*		R		D		A		A		;	

*En esta celda se realizó el cambio.

Resultados de la ejecución de la MTAR



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 MTAR.

5.3. Constructibilidad o no constructibilidad de la m-función cambiarInstrucción?

El título de esta sección "Es posible formalizar la idea de la máquina de Turing autorreferencial?" refleja la incertidumbre acerca de la posibilidad de construir una MTAR. De acuerdo a lo observado en las secciones precedentes, la posibilidad o imposibilidad de construir una MTAR se reduce a la posibilidad o imposibilidad de especificar la m-función **cambiarInstrucción**. Es decir, el núcleo de la MTAR 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**(marca1, marca2)
5. **reemplazarInstrucción**(marca1, marca2)
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, ¿ porqué suponer que es imposible ?. A manera de ejemplo se describen las m-funciones **imprimirInstrucción** y **marcarInstrucción**, adicionalmente se describe la m-función $\mathbf{PE}_n(S, \alpha_1, \alpha_2, \ldots, \alpha_n)$ que es usada por la m-función **imprimirInstrucción**.[17]

[17] Esta descripción se realiza con base en la metodología descrita en la sección 2.3.

5.3.1. Descripción m-función $\mathbf{PE}_n(\boldsymbol{S}, \alpha_1, \alpha_2, \ldots, \alpha_n)$

Explicación: A partir de las m-funciones $\mathbf{PE}(\boldsymbol{S}, \beta)$ y $\mathbf{PE}_2(\boldsymbol{S}, \alpha, \beta)$ (definidas en la sección 2.3 es posible generalizar y construir la familia de m-funciones $\mathbf{PE}(\boldsymbol{S}, \alpha)$, $\mathbf{PE}_2(\boldsymbol{S}, \alpha_1, \alpha_2)$, $\mathbf{PE}_3(\boldsymbol{S}, \alpha_1, \alpha_2, \alpha_3)$, ..., $\mathbf{PE}_n(\boldsymbol{S}, \alpha_1, \alpha_2, \ldots, \alpha_n)$; las cuales imprimen al final los símbolos α; α_1, α_2; $\alpha_1, \alpha_2, \alpha_3$; ... $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n$; respectivamente. $\longrightarrow \boldsymbol{S}$.

Descripción: Se presenta la descripción de la m-función $\mathbf{PE}_n(\boldsymbol{S}, \alpha_1, \alpha_2, \ldots, \alpha_n)$

Estado actual			Estado final
$PE_n(\boldsymbol{S}, \alpha_1, \alpha_2, \ldots, \alpha_n)$			$PE(PE((\ldots PE(PE(\boldsymbol{S}, \alpha_n), \alpha_{n_1}), \ldots), \alpha_2), \alpha_1)$

Observaciones: Técnicamente no es correcto afirmar que $\mathbf{PE}_n(\boldsymbol{S}, \alpha_1, \alpha_2, \ldots, \alpha_n)$ es una m-función. En realidad es una meta-m-función que permite representar cualquier m-función de la forma $\mathbf{PE}_n(\boldsymbol{S}, \alpha_1, \alpha_2, \ldots, \alpha_n)$. 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.

5.3.2. Descripción m-función imprimirInstrucción(S, instrucción)

Explicación: La m-función **imprimirInstrucción**($\boldsymbol{S}$, instrucción) imprime los símbolos de la cadena instrucción. $\longrightarrow \boldsymbol{S}$.

Descripción: Se presenta la descripción de la m-función **imprimirInstruccion**(S, instruccion)

Estado actual	Simb.	Comportamiento	Estado final
$\boldsymbol{imprimirInstruccion}$ $(\boldsymbol{S}, instruccion)$			$PE_n(\boldsymbol{S}, instruccion)$

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

5.3.3. Descripción m-función marcarInstrucción($\boldsymbol{S}$, α)

Explicación: La m-función **marcarInstrucción**($\boldsymbol{S}$, α) marca con el símbolo α la instrucción que está a su derecha. $\longrightarrow \boldsymbol{S}$.

Descripción Se presenta la descripción de la m-función **marcarInstruccion**($\boldsymbol{S}$, α)

Estado actual	Simb.	Comporta-miento	Estado final
$\boldsymbol{marcarInstruccion}(\boldsymbol{S}, \alpha)$	D	$R, P\alpha, R$	$marca_2$
$marca_2$	A	$R, P\alpha, R$	$marca_2$
	D	$R, P\alpha, R$	$marca_3$
$marca_3$	C	$R, P\alpha, R$	$marca_3$
	D	$R, P\alpha, R$	$marca_4$
$marca_4$	C	$R, P\alpha, R$	$marca_4$
	L	$R, P\alpha, R$	$marca_5$
	R	$R, P\alpha, R$	$marca_5$
	N	$R, P\alpha, R$	$marca_5$
$marca_5$	D	$R, P\alpha, R$	$marca_6$
$marca_6$	A	$R, P\alpha, R$	$marca_6$
	;	$R, P\alpha, R$	$\boldsymbol{S}$

6. Por qué es imposible construir una máquina de Turing autorreferencial?

Se espera que el título de esta sección sorprenda al lector. Dicha sorpresa está sustentada en el desarrollo realizado hasta el momento; en particular los resultados ofrecidos por la sección 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 MTAR.

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 mismos, considerando el proceso como los resultados, aspectos fundamentales. Este artículo hubiera sido mucho más corto si simplemente se hubiera mencionado que es una MTAR 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 desarrollo no es ilustrado, motivó la presentación realizada.

La razón de no constructibilidad de la MTAR es que la MTAR y la MUT manejan por definición diferentes códigos para representar los mismos símbolos. La MTAR debe desplazarse por la cinta en el área asignada a su descripción, para modificar los símbolos en los cuales está codificada dicha descripción, pero estos símbolos están codificados en un código diferente al utilizado por la MTAR, luego la MTAR es incapaz de modificar su descripción. Se puede afirmar que es una incapacidad **sintáctica**, la MTAR 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 MTAR debe cambiar el símbolo $''\square''$ por el símbolo $''C''$, es-

te 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 MTAR la cadena de símbolos $''DC''$ por la cadena de símbolos $''DCC''$ (esto ejemplo se ilustró en la sección 5.2.3).

Sea la siguiente codificación para los símbolos de la MTAR:

$\square$ D
0 DC
1 DCC
A $DCCC$
C $DCCCC$
D $DCCCCC$
R $DCCCCCC$
L $DCCCCCCC$
; $DCCCCCCCC$
La instrucción que cambia el símbolo $''\square''$ por el símbolo $''C''$ en la MTAR tendría la forma:

$q_x \ \square \ C \ N \ q_{x+1}$

Para efectos de la ilustración supongamos que la instrucción es:

$q_1 \ \square \ C \ N \ q_2$.

La codificación de esta instrucción para ser ejecutada por la MUT es:
$DADDCCCCNDAA$

Pero esta codificación representa los símbolos que se desean cambiar $''(''$ y $''C''$ por las secuencias de símbolos $''D''$ y $''DCCCC''$ respectivamente. Es decir, mientras que la MTAR opera con el símbolo $''C''$ la MUT opera con la secuencia $''DCCCC''$. Por esto se afirma que la MTAR 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 $''DCCC''$, $''DCCCC''$, $''DCCCCC''$, $''DCCCCCC''$, $''DCCCCCCC''$ y $''DCCCCCCCC''$, respectivamente.

7. Conclusiones, ¿ Posibles soluciones?, ¿ Nuevos problemas?

La imposibilidad de constructibilidad de la MTAR está sustentada en un problema de códigos: la MTAR es construía en un código particular y la MUT utiliza otro código para ejecutar la MTAR. Desde este punto de vista surgen dos posibles alternativas de solución.

Dado que el problema se presenta por la utilización de dos códigos diferentes, por que no pensar en seleccionar exclusivamente uno de ellos para realizar todo el trabajo [18]. Esta alternativa se relaciona con algunos lenguajes de programación (LISP por ejemplo) que codifican los datos y las instrucciones bajo el mismo código por lo cual la modificación de alguno de ellos (código o datos) se realiza de la misma forma. El autor no visualiza en este momento como seleccionar un único código para construir la MTAR (instrucciones) y para almacenarla para ser ejecutada por la MUT (datos). Lo que parece claro es que el código de las instrucciones $''q_i, s_j, s_k, m, q_l''$ no podría ser el seleccionado debido a que cada celda de la cinta (por definición) solo puede almacenar un símbolo, entonces, cómo almacenar e identificar los símbolos q_i y q_i' por ejemplo?. Esto lleva a seleccionar el código utilizado por la MUT como el código para construir la MTAR, pero es en este punto donde las cosas no están muy claras para el autor, tal como se vislumbra la situación, esto exigiría modificar completamente la definición de la MT y por extensión la de la MTAR. Además, si se toma un lenguaje como LISP como referencia, él cual tiene como base teórica al λ-cálculo y debido a que las funciones λ-definibles son coextensivas con las funciones Turing-computables, entonces, es de esperarse que lo realizable en el λ-cálculo sea realizable por medio de una máquina de Turing.

Por otra parte, si el problema es debido a la utilización de dos códigos diferentes, en particular de un código (instrucciones de la MTAR) y un meta-código (datos de la MUT), por que no seleccionar un meta-meta-código en cual codificar los dos anteriores y operar con él [19]. Inicialmente es necesario decir que aunque este meta-meta-código no está construido, no es difícil prever la posibilidad de codificar el código y el meta-código en él. La pregunta importante es: Una vez hecha está nueva codificación, que beneficios se obtendrían?. Por el momento el autor no tiene respuesta e este interrogante, pero invita al lector a reflexionar sobre él.

Con respecto al resultado obtenido, se menciona que es una confirmación adicional de la imposibilidad de ampliar la noción de computabilidad por métodos "tradicionales". No obstante, las reflexiones elaboradas a partir de la posibilidad de la construcción de la MTAR 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, la posición del autor se mantiene y refuerza en relación con: el aceptar la complejidad, la posible característica de inconsistencia de una nueva noción de computabilidad, la necesidad de modificar el substrato lógico, etc. Estas ideas servirán como directrices flexibles en la búsqueda de un nuevo significado para la computabilidad.

Se espera que la presentación de los resultados obtenidos en relación con la

[18] Esta alternativa fue sugerida por el profesor Juan Guillermo Lalinde.

[19] En conversaciones sostenidas con el profesor Rául Gómez, él se inclinó por suponer que esta alternativa produciría resultados de imposibilidad similares a los encontrados por el autor, en el desarrollo de las MTAR.

construcción de la MTAR 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. Sería muy grato conocer que en los resultados presentados existe un error o un punto de vista no identifiacdo, el cual una vez subsanado o desarrollado, permita ampliar la definición de computabilidad.

Finalmente, aunque se reconoce que la noción de computabilidad goza de un *status* muy bien establecido en la actualidad, diferentes áreas de la ciencia-filosofía acometen contra ella. Los resultados y las dificultades obtenidas en la representación de sistemas biológicos, en la construcción de máquinas cuánticas; la vigencia de conceptos como las máquinas no triviales, la auto-referencia, la recursividad; las reflexiones de (algunos) pensadores contemporáneos; las posibilidades presentes en la lógica paraconsistente; etc.; permiten prever la emergencia de una nueva definición de computabilidad. Y es precisamente, durante la reflexión acerca de y la especificación de posibles máquinas-biológicas, de posibles máquinas-cuánticas, de posibles máquinas-auto-referenciales, de posibles máquinas-paraconsistentes o para denotarlo en un solo término, de posibles máquinas-complejas; que (se espera) emerja una nueva y más potente definición de computabilidad.

Referencias

[1] George Kampis. Life-like computing beyond the machine metaphor. Eprint: `http://hps.elte.hu/Papers/`, 1992.

[2] Edgar Morin. *Introduccin al Pensamiento Complejo*. Coleccin: Ciencias Cognitivas. Editorial GEDISA, 1990.

[3] Roger Penrose. *The Emperor's new Mind Concerning Computers, Minds and the Laws of Physics*. Oxford University Press, 1989.

[4] Roger Penrose. *Las Sombras de la Mente*. Coleccin: Drakontos. Crtica, 1996.

[5] Luis M. Rocha. Artificial semantically closed objects. *Communication and Cognition - Artificial Intelligence*, 12(1–2):63–89, 1995.

[6] Hartley Rogers. *Theory of Recursive Functions and Effective Computability*. The MIT Press, 3rd edition, 1992.

[7] Wilfred Sieg. Step by recursive step: Church's analysis of effective calculability. *The Bulletin of Symbolic Logic*, 3(2):154–180, june 1997.

[8] Robert I. Soare. Computability and recursion. *The Bulletin of Symbolic Logic*, 2(3):284–321, sept. 1996.

[9] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 42:230–265, 1936-7. A correction, ibid, vol. 43, no. 2198, p. 544–546, 1937.

[10] Ludwing von Bertalanffy. *Teora General de los Sistemas*. Fondo de Cultura Econmica (FCE), 1994.

[11] Heinz von Foerster. Principios de autoorganizacin en un contexto socioadministrativo. In Marcelo Pakman, editor, *Las Semillas de la Ciberntica*, Coleccin: Terapia Familiar, page ? Gedisa, 1984.