SOBRE LA TEORÍA DE EFECTIVIDAD TIPO 2 Y ALGUNAS APLICACIONES EN EL ÁLGEBRA LINEAL

DANIEL STIVEN BERMÚDEZ ROJAS

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE CIENCIAS Y EDUCACIÓN
MATEMÁTICAS
BOGOTÁ D.C
2020

SOBRE LA TEORÍA DE EFECTIVIDAD TIPO 2 Y ALGUNAS APLICACIONES EN EL ÁLGEBRA LINEAL

DANIEL STIVEN BERMÚDEZ ROJAS

Trabajo presentado como requisito para optar al título de:
Matemático

Dirigido por:
Dra. Carolina Mejía Moreno

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE CIENCIAS Y EDUCACIÓN
MATEMÁTICAS
BOGOTÁ D.C
2020

Nota de Aceptación

_____

_____

_____

_____

_____

_____

_____

Firma del Director

_____

Firma del Jurado

Bogotá D.C, Julio 2020

Las matemáticas pueden ser definidas como
aquel tema del cual no sabemos nunca lo que decimos
ni si lo que decimos es verdadero.

Bertrand Russell

Agradecimientos

Agradezco en especial a mi mamá por darme la oportunidad de vivir, por el sacrificio que ha hecho para que llegara este momento y por el apoyo incondicional que siempre me brinda. Muchas gracias porque siempre has estado motivándome a seguir adelante a pesar de las adversidades que hemos vivido y jamás te has rendido, me has demostrado con tu gran fortaleza que se puede disfrutar de esta hermosa vida a pesar de los obstáculos. Gracias por tu amor, cariño, enseñanzas y cuidados que me has dado. Le doy gracias a Dios por haberme dado una mamá como tú y tenerte como un gran ejemplo en mi vida. Te amo, este trabajo lo hice gracias a ti y es para ti.

A mi hermana, muchas gracias porque siempre ha estado al pendiente de mí, cuidándome, protegiéndome y motivándome a salir adelante. Le agradezco estar siempre conmigo en los buenos y malos momentos.

A la Universidad Distrital Francisco José de Caldas por acogereme y permitirme ser uno de sus estudiantes, por ayudarme a crecer como persona. A a todos los profesores y profesoras que a diario luchan por transmitir sus enseñanzas; si hoy yo puedo hacer este trabajo, es gracias a ustedes y a sus enseñanzas, que me guiaron por este camino de las matemáticas.
A mis compañeros y amigos que me han soportado durante este proceso, que han hecho que esta etapa de mi vida sea menos complicada y más alegre.
A todas las personas que han creído y confiado en mí, les agradezco todo el apoyo.

Resumen

Este trabajo muestra una pequeña introducción a la Teoría de Efectividad Tipo 2, un tipo de computabilidad basado en las Máquinas de Turing de múltiples cintas unidireccionales. Con este modelo y usando una nueva definición computabilidad, podemos solucionar algunos problemas como inestabilidades numéricas que se encuentran cuando se usa la computabilidad clásica, lo que permite disminuir las serias diferencias que se observan entre los resultados de los cálculos que se hacen usando el modelo y la realidad. El objetivo principal, es aplicar este tipo de computabilidad en la resolución de algunos problemas propios del Álgebra Lineal, tales como, dada una matriz, encontrar su determinante o calcular su inversa.

Palabras clave: Máquinas de Turing, computabilidad, funciones computables, Teoría de Efectividad Tipo 2.

Abstract

This work is an introduction to Type-2 Theory of Effectivity. This theory is about some kind of computability that is based on Turing Machines with multiple one-way input tapes. This theory allows solve many problems in computational calculations like differences between real results and results obtained with classical software. The main objective in this text is to attack some problems in Linear Algebra like determinant calculus and inverse matrices.

Keywords: Turing machines, computability, computable functions, Type-2 theory of effectivity.

Índice general

Introducción

Estudiar cuáles funciones pueden ser computadas por un algoritmo es una de las principales preguntas que la Teoría de la Computabilidad ha tratado de responder. La primera noción de función computable, se conoce como el $\lambda-$Cálculo y fue desarrollada por Alonzo Church en 1936 [11]. Este es un modelo que permite expresar funciones como una $\lambda-$expresión, pero no establece ningún método para calcularlas.

El matemático Alan Turing desarrolló un modelo de computabilidad basado en la conocida Máquina de Turing [12]; de esta manera se establecio la noción de función Turing-computable, que se refiere a toda función para la cual existe una máquina de Turing que la calcula. Además, Turing demostró que una función es Turing-computable si y solo si, esta puede expresarse como una $\lambda-$expresión, por lo tanto, ambas definiciones resultan ser equivalentes y esto motivó el desarrollo de la conocida tesis de Church-Turing.

Durante los últimos años se ha desarrollado una extensa Teoría de Computabilidad y Complejidad Computacional, que llamaremos Teoría Tipo 1, la cual modela el comportamiento del mundo real para los cálculos en conjuntos discretos como los números naturales, las palabras finitas, etc. Sin embargo, se han desarrollado varias teorías de computabilidad alternas, una de estas, en la cual nos enfocaremos en este trabajo, es la Teoría de Efectividad Tipo 2 (TTE), la cual extiende la teoría ordinaria tipo 1 al caso infinito y la conecta con el análisis abstracto.

La Teoría de Efectividad Tipo 2 tiene su origen a partir de la definición de función real computable dada por Grzegorczyk en 1955 [4], que se basa en la definición de operadores computables en el conjunto ω^ω de sucesiones de números naturales. Su funcionamiento se basa en las máquinas de Turing que hacen los cálculos con palabras infinitas que representan a los números reales.

Esta teoría admite una noción de computabilidad basada en la continuidad función. Esta noción es aplicable a una gran variedad de problemas del análisis, pues sintetiza el problema de mostrar que una función real es computable ya que depende de la topología en la que se representen los números reales.

Este trabajo estará guiado por el artículo "Computability in Linear Algebra" [5] de Martin Ziegler y Vasco Brattka y se dividirá en tres capítulos. El primer capítulo mostrará resultados básicos necesarios para el entendimiento de la teoría, tales como, funciones y conjuntos recursivos, máquinas de Turing, funciones computables y máquinas de Turing equivalentes.

En el segundo capítulo se introduce la nueva Teoría de efectividad, la llamada Tipo 2. Esto permitirá generalizar la computabilidad para sucesiones de símbolos finitas o infinitas. Se introducirán representaciones estándar para los números reales y se investigarán los conceptos de computabilidad bajo estas representaciones. Se hablará de la topología de Cantor y se definirá la computabilidad de funciones reales. Se introducirán representaciones para las clases de funciones continuas $C(\mathbb{R})$ para así poder discutir acerca de su efectividad. Finalmente se mostrará que las funciones computables tipo 2 son continuas.

Por último, en el tercer capítulo se presentarán algunas aplicaciones de esta nueva computabilidad en problemas del Álgebra Lineal, mostrando que el determinante y la inversa de una matriz, entre otros, resultan ser funciones computables.

Capítulo 1

Preliminares

La Teoría de la Computabilidad, o Teoría de Recursión, se ocupa del estudio y la clasificación de las relaciones y aplicaciones computables, es decir, su propósito inicial es hacer precisa la noción intuitiva de función calculable, una función cuyos valores pueden ser calculados de forma automática o efectiva mediante un algoritmo.

En este capítulo se presentarán los conceptos básicos necesarios para entender la Teoría de la Computabilidad, empezando por los algoritmos, pasando por las funciones recursivas para llegar finalmente al concepto conocido como Máquina de Turing, ilustrando sus propiedades y variaciones.

1.1. Conceptos Básicos

Un primer concepto central en esta teoría es el de función computable. Las funciones computables son el objeto básico de estudio de la Teoría de Computabilidad y son las funciones que pueden ser calculadas por una máquina de cálculo.

Diremos además que un problema es computable cuando existe un procedimiento efectivo (algoritmo) que permite obtener, para cualquier entrada, una cadena que corresponde a una solución del problema.

Definición 1.1.1. Un **Algoritmo** es un método cuya ejecución consiste en la aplicación, paso a paso, de ciertas reglas de transformación de símbolos, especificadas a priori, las cuales deben determinar un resultado final completamente.

Cuando debemos sumar, restar, multiplicar así como resolver una división con decimales, sin darnos cuenta debemos usar un algoritmo, pues estamos haciendo una iteración de los mismos pasos continuamente. De la misma manera, pero de una forma más compleja, un programa de computador también utiliza un algoritmo para resolver problemas, luego podemos construir algoritmos que manipulen estructuras finitas de todo tipo para encontrar soluciones a diferentes prroblemas. Ahora la idea es manejar algoritmos que trabajen sobre sucesiones finitas de símbolos de un alfabeto dado.

Definición 1.1.2. Un conjunto A se dice que es **enumerable** si $A \neq \emptyset$ y si existe una función sobreyectiva $f : \mathbb{N} \to A$. En tal caso se dice que f es una *enumeración* de A, y se puede visualizar como una sucesion infinita:

$$f(0), f(1), f(2), ...$$

en la cual aparecen todos los elementos de A, con posibles repeticiones, y solo elementos de A.

Ejemplo 1.1.1. Consideremos el conjunto de los números pares $P = \{2n : n \in \mathbb{N}\}$. La aplicación $\sigma : \mathbb{N} \to P$, definida por $\sigma(n) = 2n$ para todo $n \in \mathbb{N}$, es biyectiva, luego P es equipotente a $\mathbb{N}$, es decir que P es enumerable. ⧫

Podemos imaginar algoritmos que empleen todo tipo de estructuras finitas, por ejemplo, matrices, grafos, etc. Sin embargo, no hay pérdida de generalidad cuando se restringe a algoritmos que trabajan solo sobre sucesiones finitas de símbolos en un alfabeto dado, ya que las estructuras finitas se pueden "linealizar" por medio de códigos adecuados. Vamos a notar por Σ al conjunto de símbolos (alfabeto), y Σ^* al conjunto de todas las cadenas finitas de símbolos de Σ, incluyendo la cadena vacía Λ:

$$\Sigma^* = \{\Lambda\} \cup \{s_1 \dots s_n \mid s_i \in \Sigma, n \in \mathbb{N}\}$$

A los elementos de Σ^* se les llama también *palabras* sobre Σ.

Definición 1.1.3. Un conjunto $A \subseteq \Sigma^*$ es **efectivamente enumerable** si es vacío o existe un algoritmo que genera una sucesión de elementos de A en la cual aparece eventualmente todo elemento de A.

La mayoría de los conjuntos infinitos enumerables que conocemos son efectivamente enumerables puesto que para describirlos de alguna forma utilizamos un algoritmo; de hecho el conjunto Σ^* es efectivamente numerable pues la enumeración que lista las palabras de acuerdo a su longitud puede generarse fácilmente con un programa de computador; otros conjuntos tales como el conjunto de fórmulas bien formadas también es efectivamente enumerable.

Ejemplo 1.1.2. Si p, q, r son letras proposicionales, el conjunto de fórmulas bien formadas fbf denotado por $F(p,q,r) \subseteq \{p,q,r,\neg,\wedge,\vee,\supset,(,)\}^*$ es efectivamente enumerable, pues existe un algoritmo que enumera primero las palabras de longitud $1: p,q,r$, luego las palabras de longitud $4: \neg(p), \neg(q), \neg(r)$, despues las de longitud $7: \neg(\neg(p)), (p)\wedge(q), \ldots$, y así sucesivamente. ♦

Otra noción más fuerte es la de conjunto decidible que se da para un universo enumerable, por ejemplo, las palabras sobre un alfabeto finito. Se pide que, dada una palabra, se pueda decidir algorítmicamente si esta pertenece o no al conjunto formalmente.

Definición 1.1.4. Sea Σ un alfabeto finito, un subconjunto A de Σ^* es **decidible** si existe un algoritmo que permita averiguar de una palabra arbitraria $\alpha \in \Sigma^*$ si $\alpha \in A$ o $\alpha \notin A$.

De la definición anterior podemos deducir que todo conjunto $S \subseteq \Sigma^*$ decidible es efectivamente enumerable. El conjunto de las palabras sobre Σ es efectivamente enumerable, hay un algoritmo para enumerarlas. Modificando dicho algoritmo para enumerar todas las palabras y preguntado si cada palabra producida α pertenece o no a S, se obtiene la enumeración de S. Si la respuesta es positiva α entra en la numeración de S, si no, pues no entra.

Ejemplo 1.1.3. Sea $P = \{n \in \mathbb{N} \mid \text{ n es primo}\}$ de los números primos, veamos que P es decidible, en efecto, sea $n \in \mathbb{N}$, para verificar que es primo basta con dividir a n por todos los números naturales cuyo cuadrado es menor que $n+1$; si en algún caso su residuo es cero,

entonces n no es primo y por tanto $n \notin P$; de lo contrario si en todos los casos el residuo es distinto de 0, se tiene que n es primo, es decir que $n \in P$. Este proceso se reduce a aplicar el algoritmo de la división un número finito de veces. ♦

En las definiciones anteriores se ha utilizado la noción de función computable por medio de un algoritmo; sin embargo, se hace necesario mostrar métodos para identificar funciones computables y para caracterizarlas intrínsecamente. Para esto hablaremos de las funciones recursivas. Vamos a definir estas funciones primero definiendo una subclase de la clase de funciones recursivas, la cual contiene la mayor parte de las funciones numéricas.

Definición 1.1.5. Una función $h : \mathbb{N}^k \to \mathbb{N}$ se dice **recursiva primitiva (pr)** si puede obtenerse por una sucesión finita de aplicaciones de las siguientes reglas:

I. Las siguientes funciones, llamadas *básicas*, son recursivas primitivas

$Z(x)) = 0$	función constante 0
$S(x) = x + 1$	función sucesor
$P_k^n(x_1, ..., x_n) = x_k$	k-ésima proyección en n variables, para cada $n, k \in \mathbb{N}^+$ con $k \leq n$

II. *Composición*

Si $f(y_1, ..., y_n)$ y $g_1(x_1, ..., x_k), ..., g_n(x_1, ..., x_k)$ son **pr** entonces
$h(x_1, ..., x_k) = f(g_1((x_1, ..., x_k), ..., g_n(x_1, ..., x_k))$ es **pr**

III. *Recurrencia Primitiva*

Si $f(x_1, ..., x_k)$ y $g(x_1, ..., x_k, y, z)$ son **pr**, entonces la función $h(x_1, ..., x_k, y)$ definida por

$$h(x_1, ..., x_k, y) = \begin{cases} h(x_1, ..., x_k, 0) = f(x_1, ..., x_k) \\ h(x_1, ..., x_k, S(y)) = g(x_1, ..., x_k, y, h(x_1, ..., x_k, y)) \end{cases}$$

es **pr**.

Ejemplo 1.1.4. Vamos a ver que la función factorial $x!$ es **pr**. Las ecuaciones de recursión de dicha función son:

$$0! = 1$$

$$(x+1)! = x! \cdot S(x)$$

más precisamente, si escribimos $h(x) = x!$, tenemos

$$h(0) = 1$$

$$H(t+1) = g(t, h(t))$$

donde $g(x_1, x_2) = s(x_1) \cdot x_2$; finalmente g es recursiva primitiva pues

$$g(x_1, x_2) = s(p_1^2(x_1, x_2)) \cdot p_2^2(x_1, x_2)$$

y, por tanto h, la función factorial, es recursiva primitiva **pr**. ♦

Una función $f(x_1, ..., x_k, y)$ es *regular para* y si para todo $n_1, ..., n_k$ existe m tal que $f(n_1, ..., n_k, m) = 0$. En este caso puede definirse una nueva función:

$$h(f_n) = \mu y(f(x_1, ..., x_k, y) = 0)$$

Donde $\mu y(\cdot)$ significa el "mínimo y tal que $(\cdot)$"; h se llama *minimización* de f con respecto a y. Si a las funciones **pr** le agregamos esta nueva función minimización, tenemos el conjunto de las funciones recursivas.

Definición 1.1.6. Una función f es **recursiva** si es una de las funciones básicas o puede obtenerse de las funciones básicas por un número finito de aplicaciones de composición, recurrencia primitiva y la nueva regla de minimización:

IV. *Minimización (de funciones regulares)*

Si $f(x, y)$ es recursiva y regular para y, entonces $h(x) = \mu y(f(x, y) = 0)$ es recursiva.

Con esto podemos ver que toda función recursiva primitiva es primitiva; además puesto que existe un método claro para calcular $\mu y(f(x, y) = 0)$, evaluando sucesivamente $f(x, 0), f(x, 1), ...$ hasta obtener el primer 0 donde $f(x, y)$ es calculable, obtenemos que las funciones recursivas son también calculables.

Ejemplo 1.1.5. La función $p(n) =$*n-ésimo primo* es recursiva.

Consideremos la función

$$f(x) = \begin{cases} 1 & \text{si} \quad x \text{ es primo} \\ 0 & \text{si} \quad x \text{ no es primo} \end{cases}$$

Es fácil ver que esta función es **pr** y además es recursiva. En efecto, como hay una infinidad de primos, la función

$$h(x,y) = \overline{Sg}(f(y) \cdot (y \dot{-} x)) = \begin{cases} 0 & \text{si } y \text{ primo, } y > x \\ 0 & \text{e.o.c} \end{cases}$$

es regular, donde $(x \dot{-} y)$ es la resta truncada y $\overline{Sg}$ es la función signo inversa, que se definen como

$$(x \dot{-} y) = \begin{cases} x - y & \text{si} \quad x \geq y \\ 0 & \text{si} \quad x < y \end{cases} \qquad \overline{Sg}(x) = 1 \dot{-} Sg(x) = \begin{cases} 1 & \text{si} \quad x = 0 \\ 0 & \text{si} \quad x \geq 1 \end{cases}$$

Por lo tanto la función $g(x) = \mu y(h(x,y) = 0)$ es recursiva y podemos definir por recurrencia

$$p(0) = 2$$
$$p(n+1) = g(p(n)) = \mu y(h(p(n), y) = 0) = \mu y(y \text{ es primo}, y > p(n))$$

Habiendo definido las funciones recursivas podemos decir entonces que un conjunto $S \subseteq \mathbb{N}^k$ es recursivo si su función característica

$$C_s(x_1, ..., x_k) = \begin{cases} 1 & si \quad (x_1, ..., x_n, 0) \in S \\ 0 & si \quad (x_1, ..., x_n, 0) \notin S \end{cases}$$

es recursiva.

1.2. Máquinas de Turing

Como vimos en la sección anterior toda función recursiva es calculable por medio de un algoritmo, y podríamos decir entonces que ciertos resultados se pueden realizar de forma

algorítmica, es decir que son calculables. Todo esto lo hemos logrado bajo el concepto en parte informal que tenemos de algoritmo. Sin embargo, dicho concepto falla cuando queremos mostrar que algo no es calculable y es preciso adaptar un nuevo concepto de algoritmo que contemple todos los casos posibles y que estos puedan ser analizados matemáticamente. Alan Turing propone la definición que estamos buscando en forma de autómatas capaces de calcular sobre expresiones simbólicas. De hecho, las que actualmente conocemos como Máquinas de Turing, puede considerarse, que tienen la misma capacidad que una computadora real, así este concepto haya surgido antes que se desarrollaran las computadoras. En esta sección hablaremos sobre la definición clásica de las Máquinas de Turing y cómo podemos calcular funciones recursivas mediante dichas máquinas de Turing.

Sea Σ un alfabeto finito y considere una cinta dividida en celdas que pueden estar en blanco o contener un símbolo de Σ. La cinta es potencialmente infinita, es decir, aunque no sea infinitamente larga, es posible añadirle celdas a izquierda o derecha cada vez que sea necesario.

Una máquina de Turing (MT), como vemmos en la figura 1.1, es un autómata (dispositivo que cuenta con un mecanismo interno que le permite realizar ciertos movimientos o desarrollar determinadas tareas) que consta de:

- Una **unidad de control que** opera a intervalos regulares, es decir, en pasos discretos; en cada paso realiza alguna acción, según lo especificado por cada función de transición directa.
- Una memoria auxiliar que es una **cinta infinita** con acceso relativamente no-restringido. La cinta se considera dividida en cuadrados, cada uno contiene un símbolo.
- La **cabeza de lectura/escritura** de la cinta puede moverse a lo largo de la cinta en ambas direcciones leyendo y/o escribiendo el contenido de un cuadrado uno a uno.

En palabras más resumidas una MT es un autómata que “calcula” sobre la cinta leyendo y alterando el contenido de las celdas (una a la vez) y moviendo su cabeza lectora de una a

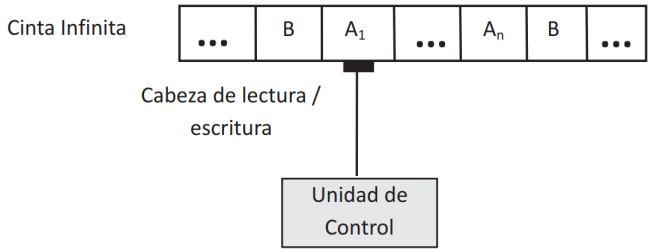


Figura 1.1: Máquina de Turing

otra celda de acuerdo con ciertas instrucciones prefijadas. La actividad de la máquina debe estar completamente determinada por el símbolo que esté leyendo y su "estado" interno en un momento dado. Para poder dar una definición más rigurosa es necesario introducir dos símbolos auxiliares llamados *movimientos*: R "derecha" y L "izquierda". El símbolo $\square$ llamado *blanco* que se supone pertence a Σ.

Definición 1.2.1. Una **Maquina de Turing (MT)** es una quíntupla $M = (K, \Sigma, M, q_0, I)$ donde:

$Q = \{q_0, q_1, q_2, ..., q_n\}$:	Conjunto finito de estados de la máquina ($Q \neq \emptyset$).
$\Sigma = \{s_0, s_1, s_2, ..., s_m\}$:	Alfabeto. Conjunto finito de símbolos entrada - salida.
	Adoptamos por convención que $s_0 = \square$ (símbolo vacío).
$M = \{L, R\}$:	Conjunto de movimientos.
q_0:	Estado inicial ($q_0 \in Q$).
I:	Es una función definida de un subconjunto de
	$Q \times \Sigma$ en $\Sigma \times M \times Q$.
	I también puede ser definida como un conjunto finito
	$I = \{i_0, i_1, i_2, ..., i_p\}$
	donde cada i_j es una quíntupla de la forma:
	$(q_m,\ s_m,\ s_n,\ m,\ q_n)$, donde
	$s_m,\ s_n \in \Sigma$; $q_m,\ q_n \in Q$; $m \in M$.
	Utilizaremos esta forma para definir el conjunto I.

La posición del cabezal y la información inicial contenida en la cinta se suministran "informalmente" antes de comenzar la ejecución de la máquina.

Si la máquina se encuentra en la situación actual (q_m, s_m) (en el estado q_m leyendo el símbolo s_m) y encuentra una instrucción $i_j : (q_m, s_m, s_n, m, q_n)$, entonces cambia el símbolo s_m por s_n, realiza el movimiento indicado por "m" (L o R) y pasa al estado q_n (puede ocurrir que $q_m = q_n$ ó $s_m = s_n$); de lo contrario, si no encuentra dicha instrucción, la máquina finaliza su ejecución.

Ejemplo 1.2.1. Definamos una máquina de Turing sobre el alfabeto $\Sigma = \{1, 0\}$ que se inicializa en una celda a la izquierda del primer caracter de una palabra escrita en la entrada de la máquina, la palabra w donde $w \in \{1^w \mid w \in \mathbb{N}\}$. La máquina debe duplicar todos los símbolos "1" que encuentre y poner 0 entre los "1" de la entrada y los "1" de trabajo. Así, si tenemos la entrada 111, devolverá 1110111; y si tenemos 1111, devolverá 111101111 y así sucesivamente.

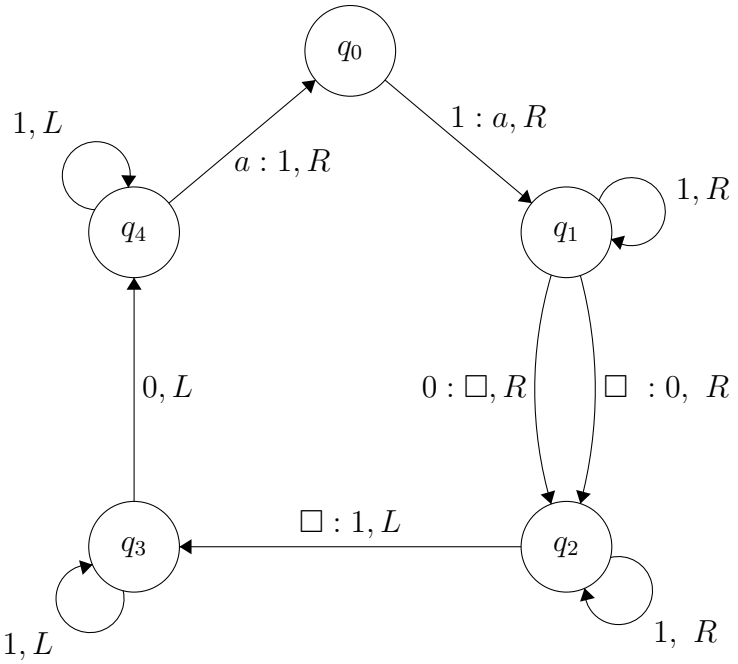
En la tabla 1.1 definimos la función I, la cual determina la máquina de Turing con estados $Q = \{q_0, q_1, q_2, q_3, q_4\}$ y estado inicial q_0, además usa el símbolo auxiliar "a".

I	0	1	a	$\square$
q_0	-	aRq_1	-	-
q_1	$0Rq_2$	$1Rq_1$	-	$0Rq_2$
q_2	-	$1Rq_2$	-	$1Lq_3$
q_3	$0Lq_4$	$1Lq_3$	-	-
q_4	-	$1Lq_4$	$1Rq_0$	-

Tabla 1.1: Máquina de estados Q

La máquina realiza su proceso por medio de un bucle; en el estado inicial q_0, reemplaza el primer 1 con una "a"; pasa al estado q_1 con el que avanza hasta la derecha, saltando los símbolos 1 hasta que encuentre un $\square$ (que debe existir); cuando lo encuentra lo cambia por

un 0 y pasa a ser q_2 (después del primer bucle, va a encontrar un 0 el cual salta y pasa a ser q_2); avanza saltando los 1 hasta encontrar otro $\square$ (la primera vez no habrá ningún 1). Una vez en el extremo derecho añade un 1. Después comienza el proceso de retorno; con q_3 vuelve a la izquierda saltando los 1; cuando encuentra el 0 (en el medio de la secuencia), pasa a q_4 que continúa a la izquierda saltando los 1 hasta el símbolo "a" que se escribió al principio. Se reemplaza de nuevo este "a" por 1, y pasa al símbolo siguiente; si es un 1, se pasa a otra iteración del bucle, pasando al Estado q_1 de nuevo. Si es un símbolo 0, será el símbolo central con lo que la máquina se detiene al haber finalizado su cómputo.
El funcionamiento de la máquina de Turing se explica mejor gráficamente por un *diagrama de transición* de estados. De este ejemplo obtenemos el diagrama:



Aquí, la flecha indica el cambio de estados; $\underrightarrow{s : t, R}$ significa: cambie el símbolo s por t y muévase a la derecha; $\underrightarrow{s, R}$ significa: muévase a la derecha (sin alterar el símbolo leído). Análogamente para $\underleftarrow{s : t, L}$ o $\underleftarrow{s, L}$, con la diferencia de que ahora el movimiento es a la izquierda. ♦

Para que una máquina de Turing realice un cálculo, se deben producir cambios en el estado actual, el actual contenido de la cinta y la ubicación del cabezal de lectura. Un establecimiento de estos tres elementos se denomina configuración de la máquina de Turing.

Definición 1.2.2. Una **configuración** de una máquina de Turing M es una expresión de la forma

$$s_1 s_2 ... s_{i-1} q s_i ... s_n$$

donde los símbolos $s_1, ..., s_n$ están en Σ y $q \in Q$. Esta expresión representa la situación actual del cómputo, es decir que la configuración $s_1 s_2 ... s_{i-1} q s_i ... s_n$ indica que la unidad de control de M está en el estado q escaneando el símbolo s_i.

El funcionamiento de la MT se representa mediante el cambio entre una configuración y otra.

1.2.1. Máquinas Equivalentes

Las máquinas de Turing pueden sufrir modificaciones las cuales no incrementan la capacidad computacional, ni tampoco modifican los lenguajes aceptados por estas; en otras palabras, los distintos modelos o variaciones de una máquina de Turing resultan ser equivalentes al modelo original. Sin embargo, estas variaciones añaden recursos computacionales que servirán para diseñar algunos algoritmos, los cuales son demasiado complejos de desarrollar en las MT estándar.

A toda máquina de Turing M sobre el alfabeto Σ se le puede asignar una función parcial

$$f_M :\subseteq \Sigma^* \longrightarrow \Sigma^*$$

Esta función se aplica cuando la máquina ha terminado de leer una palabra $w \in \Sigma^*$ impresa en la cinta, comenzando en el estado q_0 y el cabezote ubicado en el primer símbolo a la izquierda de la palabra. La función obtiene $f_M(w)$ el cual se define como la sucesión de símbolos de la configuración final que se extiende a partir de la celda donde se detuvo la máquina (puede ocurrir que la máquina con entrada $w \in \Sigma$ no se detenga, en este caso $f_M(w)$ queda indefinida).

El dominio de dicha función f_M es el conjunto D definido de la siguiente manera

$$D := \{w \in \Sigma^* : f_M(w) \text{ está definida}\}$$

Habiendo definido esta función podemos decir que dos máquinas de Turing M y M' son equivalentes si $f_M = f_{M'}$, es decir, si las dos funciones tienen el mismo dominio D, y si $w \in D$, entonces $f_M(w) = f_{M'}(w)$.

Observando la definición anterior, vemos que no hay pérdida de generalidad si las máquinas trabajan solo en la parte derecha de la cinta, es decir, en la semicinta que tiene una celda a la izquierda de la posición inicial y se extiende solo a la derecha.

Lema 1.2.1. *Para toda máquina de Turing M existe una máquina equivalente M' que nunca visita las celdas a la izquierda de la celda anterior a la inicial.*

La demostración de este lema se puede ver de manera detallada en [1].

Como vimos con el lema anterior, podemos encontrar variantes de las máquinas de Turing. Existen, por ejemplo, máquinas de Turing multipista, máquinas de Turing no deterministas, entre otras. Sin embargo, nuestro interés son las máquinas de Turing multicinta.

Una máquina de Turing multicinta es como una máquina de Turing ordinaria pero con k cintas diferentes, cada una dividida en celdas o casillas.

Inicialmente la entrada aparece en la primera cinta y las otras cintas están llenas de blancos. En un paso computacional, la unidad de control cambia el contenido de la casilla escaneada en cada cinta y realiza luego uno de los desplazamientos R o L. Esto se hace de manera independiente en cada cinta; la unidad de control tiene entonces k "visores" o "cabezales" que actúan independientemente en cada cinta. La función de transición es cambiada para permitir la lectura, escritura y movimiento de los índices en todas las cintas simultánemente. Si k es el número de cintas, la función formalmente es:

$$I : Q \times \Sigma^k \longrightarrow Q \times (\Sigma \times M)^k$$

$$I(q_i, (s_1, ..., s_k)) = (q_j, (b_1, M), (b_2, M), (b_3, M)..., (b_k, M))$$

Donde los s_i y b_i son símbolos del alfabeto Σ y M denota un desplazamiento (L o R). La función de transición nos indica que si la máquina está en el estado q_i y los "cabezales" 1 a k leen los símbolos s_1 hasta s_k, la máquina pasa al estado q_j, escribe b_1 a b_k en las cintas 1 a k respectivamente, y mueve cada cabezal a la izquierda o a la derecha según lo especificado por M.

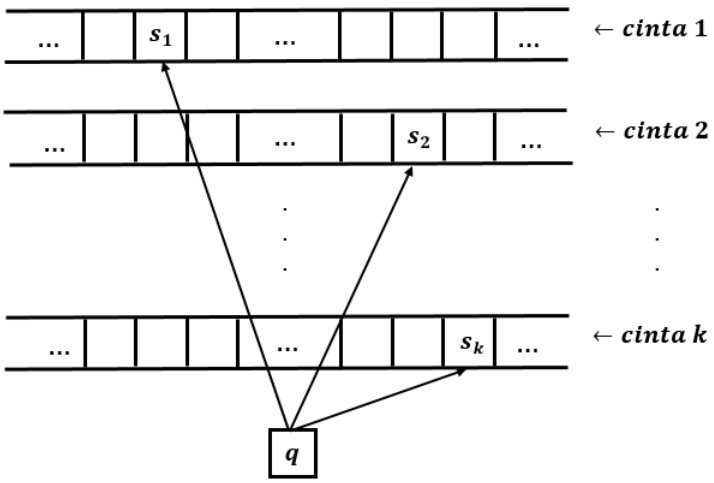


Figura 1.2: Máquina de Turing Multicinta

Ejemplo 1.2.2. Vamos a considerar una máquina de dos cintas con

$$\Sigma = \{a, b, c\}$$

$$Q = \{q_0, q_1, q_2\}$$

$$M = \{R, N, L\}$$

donde N es el movimiento neutro, que acepte el lenguaje:

$$L = \{a^i b^i c^i : i \geq 0\}$$

Es decir, que reconozca si una palabra $w \in \Sigma$ es de la forma $a^i b^i c^i$ con $i \geq 0$. Esto es bastante laborioso en una máquina de Turing con una única cinta, es mucho más fácil realizarlo con una máquina de Turing de dos cintas.

Se pone la cadena de entrada en la primera cinta; la idea es copiar en la segunda cinta una X (usado como símbolo auxiliar) por cada a, y cuando encuentre la primera b, se detiene en la primea cinta, luego se avanza a la derecha en la primera cinta y se avanza a la izquierda en la segunda cinta; cuando encuentra la primera c las dos cintas avanzan hacia la derecha. La función de transición I es la siguiente:

$$\begin{aligned}
I(q_0, (a, \square)) &= (q_0, (a, X), (R, R)) \\
I(q_0, (b, \square)) &= (q_1, (b, \square), (N, R)) \\
I(q_1, (b, X)) &= (q_1, (b, X), (R, L)) \\
I(q_1, (c, \square)) &= (q_2, (c, \square), (N, R)) \\
I(q_2, (c, X)) &= (q_2, (c, X), (R, R)) \\
I(q_2, (\square, \square)) &= (q_3, (\square, \square), (R, R))
\end{aligned}$$

♦

Aunque las máquinas de Turing multicinta parecen ser más poderosas que las máquinas ordinarias, con el siguiente teorema vamos a ver que las dos máquinas resultan ser equivalentes.

Teorema 1.2.1. *Toda máquina de Turing multicinta tiene un equivalente con una máquina de Turing de una cinta.*

Demostración. Vamos a mostrar como convertir una máquina de múltiples cintas M en una máquina de una sola cinta S. La idea clave es mostrar cómo simular M con S.

Supongamos que M tiene k cintas y que

- La máquina S simula el efecto de k cintas al almacenar su información en su única cinta.
- La máquina S usa un nuevo símbolo (#) como delimitador para separar el contenido de las diferentes cintas.

– La máquina S realiza un seguimiento de la ubicación de las cabezas, marcando con un • los símbolos donde estarían las cabezas.

La figura 1.3 muestra un ejemplo de como se representa una máquina M con 3 cintas por una máquina S de una cinta.

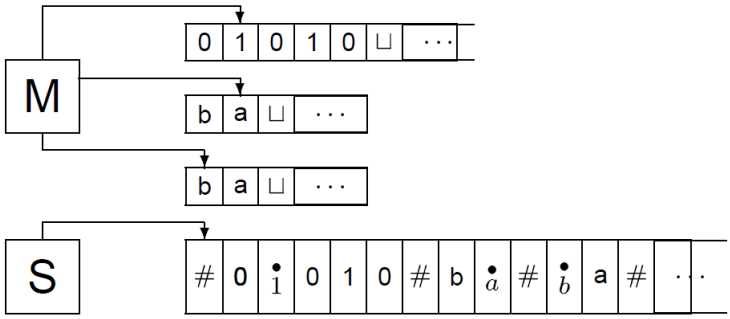


Figura 1.3: Simulación de una máquina multicinta a una máquina de una sola cinta

Ahora tomemos a S con la entrada $w = w_1w_2...w_n$ y la simulación será la siguiente:

1. La máquina S pone su cinta en el formato que representa las k cintas de M

$$S = \#\dot{w}_1...w_n\#\dot{\square}\#...\#\dot{\square}\#$$

2. Se escanea la cinta de S desde el primer #(que representa el extremo izquierdo) hasta el $(k+1)$-ésimo # (que representa el extremo derecho) para determinar los símbolos bajo los cabezales; luego S realiza un segundo escaneo sobre la cinta para actualizarla de acuerdo con la forma en que lo dicta la función de transición de M.

3. Si en algún momento S mueve uno de los cabezales a la derecha sobre un símbolo # significa que M se ha movido en la cinta correspondiente a la parte en blanco no leída de esa cinta. Entonces S desplaza el contenido de la cinta desde esta celda hasta el # más a la derecha, y escribe un □ en su cinta. Luego continua simulando como antes.

□

1.2.2. Computabilidad de Funciones recursivas

Una de las ventajas de las Máquinas de Turing es que calculan funciones recursivas y aceptan elementos de conjuntos recursivos para hacer sus cálculos.

Definición 1.2.3. Una función f es T-calculable si existe una máquina de Turing M que calcula dicha función.

El éxito de las máquinas de Turing está en la última definición que sugiere que estas máquinas pueden computar cualquier problema, representable mediante un algoritmo computable; y se puede concluir entonces, que un algoritmo es (Turing-)computable, si puede ser computado por una Máquina de Turing.

Teorema 1.2.2. *Toda función es recursiva si y solo si es T-calculable.*

La demostración del teorema la podemos ver en [1] donde se prueba en detalle que la función constante, la función sucesor, la proyección y las operaciones composición, recurrencia y minimización son T-calculables, exponiendo las máquinas de Turing que las calculan. Además se prueba que un conjunto es recursivo si su función característica es recursiva. Así podemos decir que un conjunto recursivo tambien es T-calculable.

De acuerdo con lo tratado en el presente capítulo podemos evidenciar que diseñar una MT es similar a escribir un programa computacional, puesto que la función de transición de una MT es básicamente un conjunto de instrucciones. Entonces podríamos concluir que existe una conexión directa entre las máquinas de Turing y los algoritmos. La declaración conocida como "tesis de Church-Turing" afirma que esta conexión es en realidad una equivalencia.

Tesis de Church-Turing: *Todo Algoritmo puede ser descrito por medio de una máquina de Turing.*

Este "resultado" abarca tanto los algoritmos que presentan una salida, como aquellos que nunca terminan. La tesis de Church-Turing en teoría no es demostrable pues involucra

una noción intuitiva de algoritmo. Se puede refutar encontrando un procedimiento que sea aceptado como un algoritmo y que no pueda ser descrito por una máquina de Turing, sin embargo, la experiencia corrobora cada vez más la tesis de Church-Turing.

Un hecho que permite una mayor aceptación de la Tesis de Church-Turing es la posibilidad de adicionar recursos computacionales a las máquinas de Turing (por ejemplo, las máquinas multicinta) los cuales no incrementan el poder del modelo original. Esto nos muestra que, las máquinas de Turing aparte de ser muy flexibles también representan el límite de lo que un dispositivo computacional puede hacer.

Capítulo 2

Teoría de Efectividad Tipo 2

El análisis computable es una conexión entre el análisis y la computabilidad combinando adecuadamente los conceptos de aproximación y computación. Durante mucho tiempo, se han formulado modelos de computación no equivalentes sobre los números reales. El más aceptado hasta el momento, gracias a su flexibilidad, es la Teoría por Representaciones: Teoría de Efectividad Tipo 2 (TTE por sus siglas en inglés). La idea central de esta teoría está inspirada en la definición de función real computable: *una función real es computable, si y solo si, puede ser representada como un operador computable sobre una codificación de los números reales*, formulada por Grzegorczyk [4].

En este capítulo vamos a presentar una introducción a la Teoría de Efectividad Tipo 2 TTE, unificando definiciones y notaciones disponibles en la literatura.

2.1. Máquinas Tipo 2

En el capítulo anterior vimos que, tradicionalmente, la computbilidad dado un alfabeto Σ es introducida mediante funciones parciales

$$f :\subseteq (\Sigma^*)^n \longrightarrow \Sigma^*$$

Con el objetivo de introducir nociones de computabilidad para funciones sobre otros conjuntos enumerables (tales como los naturales $\mathbb{N}$ o racionales $\mathbb{Q}$), las sucesiones finitas o

elementos de Σ^* son utilizadas como nombres. Entonces una máquina de Turing que calcula una función, transforma nombres en nombres, y el usuario es quien hace la interpretación del resultado.

Ahora bien, dado que el alfabeto Σ es finito entonces el conjunto Σ^* de sucesiones finitas es enumerable, luego este método de asociar nombres finitos a los objetos no puede ser aplicado a funciones sobre conjuntos tales como $\mathbb{R}$ o $\wp(\mathbb{R})$, los cuales no son enumerables. Por tanto, se hace necesario introducir un nuevo tipo de computabilidad, fundamentado en las máquinas de Turing, en donde las cadenas de entrada y salida sean elementos de $\Sigma^w := \{p \mid p : \mathbb{N} \to \Sigma\} = \{a_0a_1a_2 \ldots \mid a_i \in \Sigma\}$ (palabras infinitas sobre Σ) y cuyo objetivo principal sea asociar a los objetos de un conjunto infinito no enumerable una secuencia infinita de símbolos tomados de un alfabeto Σ.

En esta sección ampliaremos los conceptos vistos anteriormente, pero ahora introducimos el uso de sucesiones infinitas de símbolos como nombres, y definimos la computabilidad para las funciones aplicadas sobre tales sucesiones infinitas.

Para $k > 0$ y $Y_0, Y_1, ..., y_k \in \{\Sigma^*, \Sigma^w\}$ definimos las funciones computables

$$f :\subseteq Y_1 \times ... \times Y_k \longrightarrow Y_0$$

por medio de máquinas de Turing con k cintas de entradas unidireccionales, un número finito de cintas de trabajo y una sola cinta de salida unidireccional; esta máquina se puede ver gráficamente en la figura 2.1.

Este tipo de máquinas de Turing están dadas por un alfabeto de entrada y salida Σ con un símbolo especial $\square \notin \Sigma$, un alfabeto de trabajo Γ con $\Sigma \cup \{\square\} \subseteq \Gamma$ y un número de k cintas de entrada. La máquina opera en la cinta de salida 0, las k cintas de entrada $1, ..., k$ y las cintas de trabajo $k+1, ..., N$ (para algún número N). Además en las cintas de entrada y en la cinta de trabajo, no está permitido moverse a la izquierda y tampoco sobrescribir símbolos, esto garantiza que las cintas de entrada son citas unidireccionales de sólo lectura

y, que en la cinta de salida, sólo puedan ser escritos símbolos de Σ y que los símbolos ya escritos no puedan ser borrados (salida unidireccional).

Definición 2.1.1. ***(Máquinas Tipo 2)***
Una máquina Tipo 2 está definida por dos componentes (definición tomada de [3]):

(I) Una máquina de Turing con k cintas de entrada ($k \geq 0$), una única cinta de salida unidireccional y finitas cintas de trabajo.

(II) Un tipo de especificación $(Y_1, ..., Y_k, Y_0)$ con $\{Y_0, ..., Y_k\} \subseteq \{\Sigma^*, \Sigma^w\}$.

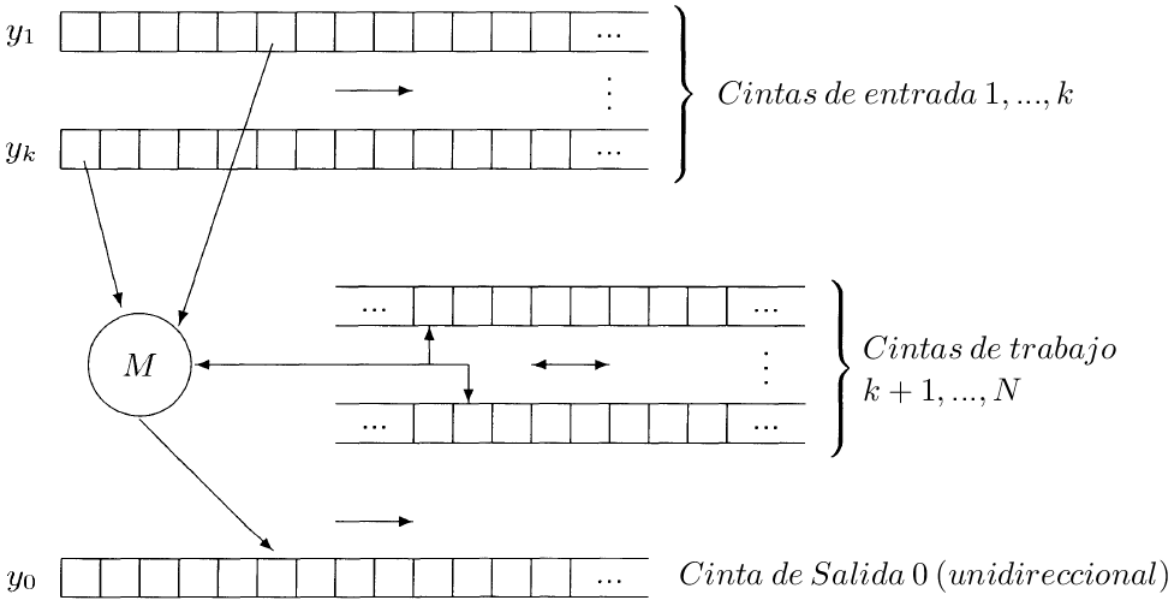


Figura 2.1: Una máquina de Turing Computando $y_0 = f_M(y_1, ..., y_k)$

El tipo de especificación expresa que $f_M :\subseteq Y_1 \times ... \times Y_k \longrightarrow Y_0$ es el tipo de las funciones computadas por una máquina M. Indica cuál de las cintas (que deben ser unidireccionales) de entrada y salida se proporcionan para sucesiones finitas y cuales para sucesiones infinitas.

Definición 2.1.2. La función $f_M :\subseteq Y_1 \times ... \times Y_k \longrightarrow Y_0$ computada por la máquina tipo 2 M es definida como sigue:

- Caso $Y_0 = \Sigma^*$ (salida finita):
  $f_M(y_1, ..., y_k) = w \in \Sigma^*$ si y solo si M con entrada $(y_1, ..., y_k)$ se detiene con el resultado w en la cinta de salida.

- Caso $Y_0 = \Sigma^w$ (salida infinita):
  $f_M(y_1, ..., y_k) = p \in \Sigma^w$ si y solo si M con entrada $(y_1, ..., y_k)$ computa para siempre escribiendo la sucesión p en la cinta de salida.

Se debe tener en cuenta que, en el caso $Y_0 = \Sigma^*$ el resultado $f_M(y_1, ..., y_k)$ es indefinido, pues la máquina sólo escribe finitos símbolos y por lo tanto nunca para.

Así como, en el capítulo anterior, definimos la computabilidad de funciones con dominio enumerable usando las máquinas de Turing, también podemos definir, con las máquinas tipo 2, la computabilidad para funciones $f :\subseteq Y_1 \times \cdots \times Y_k \longrightarrow Y_0(\{Y_0, \ldots, Y_k\} \subseteq \{\Sigma^*, \Sigma^w\})$.

Definición 2.1.3. ***(Computabilidad tipo 2)***
Sea Σ un alfabeto finito. Asuma que $\{Y_0, \ldots, Y_k\} \subseteq \{\Sigma^*, \Sigma^w\}$ $(k \geq 0)$. Una función $f :\subseteq Y_1 \times \cdots \times Y_k \longrightarrow Y_0$ es computable si y solo si $f = f_M$ para alguna máquina **M** tipo 2 (definición tomada de [2]).

Puesto que, las máquinas Tipo 2 en la práctica son tan potentes como las máquinas de Turing, las entradas y salidas infinitas no existen, ni tampoco los cálculos infinitos; sin embargo cualquier parte finita de la salida puede obtenerse de una parte inicial finita del cálculo posiblemente infinito. Es decir, el comportamiento de una máquina Tipo 2 se puede aproximar de forma adecuada en el finito, por tanto las máquinas Tipo 2 y sus cálculos son físicamente computables.

La definición anterior es la definición estándar de funciones computables. Definimos ahora los elementos computables de Σ^* y Σ^w.

Definición 2.1.4. ***(Elementos computables)***

1. Toda palabra $w \in \Sigma^*$ es computable.

2. Una sucesión $p \in \Sigma^w$ es computable, si y solo si la función constante

$$f : \{()\} \longrightarrow \Sigma^w, \qquad f() = p$$

es computable.

3. Una tupla $(y_1, y_2, \ldots, y_k)$ con $y_i \in \Sigma^w$ es computable, si y solo si cada componente y_i es computable.

Ejemplo 2.1.1. Vamos a ilustrar las definiciones anteriores con algunos ejemplos:

1. Una función constante

$$f_c : Y_1 \times \cdots \times Y_k \longrightarrow Y_0$$

con valor constante c es computable, si y solo si, el valor $c \in Y_0$ es computable.

2. Sea $f :\subseteq \Sigma^w \longrightarrow \Sigma^*$, con $\{0, 1\} \subseteq \Sigma$, definida por

$$f(p) := \begin{cases} 1 & \text{si} \quad p \neq 0^w \\ div & \text{e.o.c} \end{cases}$$

donde $f(a) = div$ si $a \notin dom(f)$. Entonces existe una máquina M Tipo 2 que lee la entrada $a_0a_1... \in \Sigma^w$, se detiene tan pronto se encuentra alguna i tal que $a_i \neq 0$ y escribe 1.

3. La función $f :\subseteq \Sigma^w \longrightarrow \Sigma^*$ con $\Sigma := \{0, 1\}$ definida por

$$f(p) := \begin{cases} 1 & \text{si} \quad p \neq 0^w \\ 0 & \text{e.o.c} \end{cases}$$

no es computable. En efecto, supongamos que si es computable entonces existe una máquina M, Tipo 2, que computa a f. Consideremos la entrada $p := 00\ldots = 0^w$, entonces para algún número $k \in w$, M produce la salida 0 en k pasos. Ahora considere la entrada $p' := 0^k10^w$, ya que los primeros k símbolos de p y p' coinciden, y ya que M puede leer en k pasos al menos k símbolos, M se detiene con la salida 0 también para la entrada p' pero $f(p') = 1$, luego M no puede calcular la función. ♦

En el ejemplo 2.1.1.3 se ha utilizado la **propiedad de finitud** para funciones computables, la cual nos dice que si w es cualquier prefijo finito de la salida $f(z) = y$ entonces w está determinado por alguna porción finita de la entrada z. Dicha propiedad termina siendo equivalente a la continuidad respecto a las topologías estándar utilizadas en la Teoría de Efectividad Tipo 2 las cuales se definen a continuación:

Definición 2.1.5. ***(Topología de Cantor en Σ^ω, Topología discreta en Σ^*)*** Ahora recordamos la definición de dos topologías ampliamente conocidas, la Topología de Cantor y la Topología Discreta. Para mayor información se puede consultar [2].

1. $\tau_d := 2^{\Sigma^*} = \{A \mid A \subseteq \Sigma^*\}$ es llamada la topología discreta en Σ^*.
2. $\tau_C := \{A\Sigma^\omega \mid A \subseteq \Sigma^*\}$ es llamada la topología de Cantor en Σ^ω, (Σ^ω, τ_C) es llamado el espacio de Cantor (sobre Σ).

Como bases canónicas para τ_d y τ_C se utilizan $\{\{w\} \mid w \in \Sigma^*\}$ y $\{w\Sigma^\omega \mid w \in \Sigma^*\}$ respectivamente.

Todo conjunto $A \subseteq \Sigma^*$ es τ_d-abierto, es decir $A \in \tau_d$. Un conjunto $U \subseteq \Sigma^\omega$ es τ_C-abierto, es decir $U \in \tau_C$, si y solo si, existe algún $A \subseteq \Sigma^*$ con $(p \in U \Leftrightarrow (\exists w \in A)$ con w prefijo de $p)$ para todo $p \in \Sigma^\omega$. Un subconjunto $X \subseteq \Sigma^\omega$ es una bola abierta y cerrada, si y solo si, el conjunto X es de la forma $X = w\Sigma^\omega$ para alguna palabra $w \in \Sigma^*$ donde $w\Sigma^\omega = B(w0^\omega, 2 \cdot 2^{-n}) = \overline{B}(w0^\omega, 2^{-n})$ con $n = long(w)$; ademas, $A\Sigma^\omega := \cup\{w\Sigma^\omega \mid w \in A\}$.

Tomando en cuenta estas definiciones podemos reescribir la propiedad de finitud para funciones $f :\subseteq \Sigma^\omega \longrightarrow \Sigma^\omega$. Supongamos que $f(z) = y$. Entonces para cualquier bola $B(y, \varepsilon)$ existe una bola $B(z, \delta)$ tal que $f(B(z, \delta)) \subseteq B(y, \varepsilon)$. Pero esto significa que f es continua en z, es decir, la propiedad de finitud es equivalente a la continuidad.

Teorema 2.1.1. *Toda función computable $f :\subseteq Y_1 \times Y_2 \times \cdots \times Y_k \longrightarrow Y_0$ por una máquina de Turing Tipo 2 es continua.*

Demostración. Sea $f(y_1, \ldots, y_k)$. consideremos el caso donde $Y_0 = \Sigma^\omega$. Para mostrar lo deseado es suficiente mostrar que para cualquier vecindad $w_0\Sigma^\omega$ de y_0 existe alguna vecindad

X de $(y_1, \ldots, y_k)$ con $f(X) \subseteq w_0\Sigma\omega$.
Sea M una máquina tipo 2 que calcula f, sea $w_0\Sigma^\omega$ una vecindad de y_0. Entonces M con la entrada $(y_1, \ldots, y_k)$ escribe el prefijo w_0 de y_0 en una cantidad finita de pasos. Durante este cálculo solo puede ser leído el prefijo w_i de la entrada y_i en la cinta i con $i = 1, \ldots, k$; entonces $X := w_1Y_1 \times \ldots \times w_kY_k$ es una vecindad abierta de $(y_1, \ldots y_k$ con $f(X) \subseteq w_0\Sigma^\omega$
Para el caso donde $Y_0 = \Sigma^*$ se procede de forma análoga. □

En el ejemplo 2.1.1.3 vimos que f no era computable pues la función presenta una discontinuidad; podríamos decir entonces que para funciones en Σ^* y Σ^w la continuidad es una condición necesaria pues sólo funciones continuas son computables, sin embargo existen funciones continuas que no son computables.

Ejemplo 2.1.2. Sea $d : \omega \longrightarrow \omega$ una función total con $rango(d) \subseteq \{0, 1\}$ que no es computable. Entonces las funciones $f :\subseteq \Sigma^* \longrightarrow \Sigma^\omega$, $g :\subseteq \Sigma^\omega \longrightarrow \Sigma^*$ y $h :\subseteq \Sigma^\omega \longrightarrow \Sigma^\omega$, son continuas pero no son computables, donde:

$$f(w)(n) := d(n) \qquad \forall w \in \Sigma^*, n \in \omega$$

$$g(0^\omega) := div$$

$$f(0^k1q)(n) := d(k) \qquad \forall k \in \omega, q \in \Sigma^\omega$$

$$h(q)(n) := d(n) \qquad \forall q \in \Sigma^\omega, n \in \omega$$

♦

Otro resultado de gran importancia es la composición de funciones computables, que resultan también ser computables, y, como consecuencia, obtenemos que las funciones computables envían elementos computables a elementos computables. Este resultado se ilustra en el siguiente teorema (por simplicidad consideraremos sólo funciones unarias).

Teorema 2.1.2. *Sean $f :\subseteq Y_1 \longrightarrow Y_2$ y $g :\subseteq Y_2 \longrightarrow Y_3$ con $Y_1, Y_2, Y_3 \in \{\Sigma^*, \Sigma^w\}$ funciones computables. Si $(Y_2, Y_3) \neq (\Sigma^w, \Sigma^*)$ entonces $g \circ f$ es computable.*

Demostración. Sean M_f y M_g máquinas Tipo 2 que computan f y g respectivamente. De M_f y M_g podemos construir una máquina M Tipo 2 que simula de forma alternada los cálculos de M_f y M_g tomando a su vez los símbolos de salida de M_f como símbolos de entrada para M_g; entonces M_g se simula hasta que se requiere el primer símbolo de entrada que debe producir M_f; luego M_f empieza la simulación y produce el primer símbolo de salida, el cual toma M_g como símbolo de entrada y empieza otra vez a simular hasta que requiere el siguiente símbolo de entrada que le debe proporcionar M_f, y así sucesivamente. □

Así como vimos en el capítulo anterior, existen definiciones de recursividad y decidibilidad sobre la computabilidad clásica, ahora estos conceptos tienen su definicón equivalente en la computabilidad Tipo 2.

Definición 2.1.6. Sea $k \geq 0$ y $Y_1, \ldots, Y_k \in \{\Sigma^*, \Sigma^w\}$

1. Un conjunto $X \subseteq Y_1 \times \cdots \times Y_k$ es llamado **recursivamente enumerable (r.e.)**, si y solo si, $X = dom(f)$ para alguna función computable Tipo 2 con $f :\subseteq Y_1 \times \cdots \times Y_k \longrightarrow \Sigma^*$.

2. Para $U \subseteq W \subseteq Y_1 \times \cdots \times Y_k$ decimos que U es **r.e. en** W, si y solo si, $U = W \cap X$ para algún conjunto X r.e.

3. Para $U \subseteq W \subseteq Y_1 \times \cdots \times Y_k$ decimos que U es **decidible** en W, si y solo si, U y $W - U$ son r.e. en W.

Hasta ahora hemos definido la computabilidad en los conjuntos Σ^* de palabras infinitas y Σ^w de sucesiones infinitas explícitamente por máquinas Tipo 2. En TTE se utilizan secuencias infinitas de símbolos como nombres de objetos, y se introduce una noción de computabilidad en la que se transforman secuencias infinitas en secuencias infinitas y el usuario de la máquina interpreta estas secuencias como nombres finitos o infinitos de objetos abstractos.

Definición 2.1.7. ***(Sistemas de nombres)***
Un **Sistema de nombres** de un conjunto M es una notación o una representación de M, donde una notación es una función sobreyectiva

$$\nu :\subseteq \Sigma^* \longrightarrow M$$

y una representación es una función sobreyectiva

$$\delta :\subseteq \Sigma^w \longrightarrow M$$

En general, se escribe ν_w para abreviar $\nu(w)$ y δ_p para $\delta(p)$. Además, si $\gamma :\subseteq Y \longrightarrow M$ es un sistema de nombres (con $Y \in \{\Sigma^*, \Sigma^\omega\}$, $x \in M$), $p \in Y$ y $\gamma(p) = x$, entonces se dice que p es un $\gamma-$*nombre* de x (definición tomada de [2]).

Ejemplo 2.1.3. Algunos ejemplos clásicos de notaciones y representaciones son:

- La identidad $id_{\Sigma^*} : \Sigma^* \longrightarrow \Sigma^*$ es una notación de Σ^* y la identidad $id_{\Sigma^w} : \Sigma^w \longrightarrow \Sigma^w$
- La notación binaria de números naturales la veremos como

  $$\nu_{bin} :\subseteq \Sigma^* \longrightarrow \mathbb{N}$$

  y la representación decimal de números reales la veremos como

  $$\nu_{dec} :\subseteq \Sigma^w \longrightarrow \mathbb{R}$$ ♦

Un sistema de nombres $\gamma :\subseteq Y \to M$ transforma los conceptos vistos en la definición 2.1.6 de Y a M.

Definición 2.1.8. Sean $\gamma_i :\subseteq Y_i \longrightarrow M_i$ $(i = 1, \ldots, k)$ k sistemas de nombres, se tiene que

- $X \in M_1$ es $\gamma - 1 -$ *computable*, si existe un elemento computable $y \in Y_1$ con $\gamma_1(y) = x$.
- $X \subseteq M_1 \times \cdots \times M_k$ es $(\gamma_1, \ldots, \gamma_k)-$abierto (-r.e, -recursivo) si y solo si $\{(y_1, \ldots, y_k) \in Y_1 \times \cdots \times |(\gamma_1(y_1), \ldots, \gamma_k(y_k)) \in X\}$ es abierto (r.e., recursivo) en $dom(\gamma_1) \times \cdots \times dom(\gamma_k)$.

Ejemplo 2.1.4.

- Sea $\nu_{bin} :\subseteq \Sigma^* \longrightarrow w$ la notación binaria de w. Cada $n \in w$ es $\nu_{bin}-$computable; cada subconjunto $A \subseteq w$ es $\nu_{bin}-$abierto. Adremás tenemos que un subconjunto $A \subseteq w$ es $\nu_{bin}-$r.e., si y solo si, este es r.e.

- Sea δ_{dec} la representación decimal de números reales por fracciones decimales infinitas, entonces todo número racional es $\delta_{dec}-$computable, en efecto, notemos que los nombres decimales de números racionales son periódicos. ♦

Los conceptos de computabilidad vistos anteriormente inducidos en conjuntos por sistemas de nombres, no cambian si los sistemas de nombres son reemplazados por otros equivalentes, de igual manera pasa con propiedades como la continuidad de funciones computables. Esto se obtiene por el hecho de que las funciones computables y continuas son cerradas bajo la composición.

Todo lo visto anteriormente nos lleva a concluir que en TTE la computabilidad de un conjunto M se introduce en dos pasos:

1. La definición de funciones computables en sucesiones finitas e infinitas de símbolos .
2. La definición de un sistema de nombres $\gamma :\subseteq Y \longrightarrow M$.

En el primer paso (que es independiente del conjunto M), elegimos funciones computables de Tipo 2 que ya eran computables o "efectivas" en la teoría clásica, es decir, elegimos funciones definidas sobre sucesiones de símbolos que ya eran computables según otros criterios; en el segundo paso, el sistema de nombres del conjunto M puede definirse solo en relación con alguna estructura contenida en el conjunto M.

2.2. Computabilidad en los números reales

En esta sección vamos a exhibir sistemas de nombres para los conjuntos de los números $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$ y a estudiar la computabilidad inducida por sistemas de nombres.

Sea Σ un alfabeto suficientemente grande, sea $\nu_{\mathbb{N}} :\subseteq \Sigma^* \longrightarrow \omega$ la representación de los números naturales en binario y sea $\nu_{\mathbb{Q}} :\subseteq \Sigma^* \longrightarrow \omega$ la representación de los números racionales como fracciones irreducibles de dos enteros. Estás representaciones y todas las

equivalentes se denominan efectivas, y se abreviaran de la siguiente manera:

$$\overline{u} := \nu_{\mathbb{N}}(u) \qquad \forall u \in \mathbf{dom}(\nu_{\mathbb{N}})$$

$$\overline{u} := \nu_{\mathbb{Q}}(u) \qquad \forall u \in \mathbf{dom}(\nu_{\mathbb{Q}})$$

Con estas representaciones podemos ver que en ω la suma es computable definiéndola respecto a $\nu_{\mathbb{N}}$ como:

$$f : \omega^2 \longrightarrow \omega \qquad (x, y) \longrightarrow f(x, y) = x + y$$

Esta función f es $(\nu_{\mathbb{N}}, \nu_{\mathbb{N}}, \nu_{\mathbb{N}})-$computable, en efecto, pues existe una función computable g definida como:

$$g :\subseteq \Sigma^* \times \Sigma^* \longrightarrow \Sigma^* \qquad \text{con } f(\overline{u}, \overline{v}) = \nu_{\mathbb{N}}(g(u, v)) \quad \forall u, v \in \mathbf{dom}(\nu_{\mathbb{N}})$$

Además, la multiplicación, potenciación, división de números naturales y racionales también resultan ser computables respecto a $\nu_{\mathbb{N}}$ y $\nu_{\mathbb{Q}}$ respectivamente y lo podemos ver en detalle en [3].

La representación más popular para los números reales ($\mathbb{R}$) es la representación por fracciones decimales:

$$\delta_{dec} :\subseteq \Sigma^w \longrightarrow \mathbb{R}$$

Sin embargo, funciones como $x \longrightarrow 3x$ no son $(\delta_{dec}, \delta_{dec})-$computables pues, como se puede apreciar en [2], no existe una máquina Tipo 2 que multiplique fracciones decimales infinitas por 3, lo que implica que esta representación no proporciona elementos suficientes para la computabilidad en $\mathbb{R}$ ya que la multiplicación de números reales debe ser computable; es por esto que se hace necesario buscar otras representaciones estándar de los números reales.

Definición 2.2.1. Para los números reales se define $\rho_I :\subseteq \Sigma^w \longrightarrow \mathbb{R}$(representación por intervalos) y $\rho_C :\subseteq \Sigma^w \longrightarrow \mathbb{R}$ (representación de Cauchy) de la siguiente manera:

- $\rho_I(p) = x$, si y solo si, existen $u_0, v_0, u_1, v_1, \ldots \in \mathbf{dom}(\nu_{\mathbb{Q}})$ con $p = u_0\#v_0\#u_1\#v_1 \ldots$ y

$$x = \sup_{i \in \omega} \overline{u_i} = \inf_{i \in \omega} \overline{v_i}$$

- $\rho_C(p) = x$, si y solo si, existen $u_0, u_1, \ldots \in \mathbf{dom}(\nu_{\mathbb{Q}})$ con $p = u_0\#u_1 \ldots, (\forall k)(\forall i > k)$ $\mid \overline{u_i} - \overline{u_k} \mid < 2^{-k}$ y

$$x = \lim_{k\to\infty} \overline{u_k}$$

Si $p = u_0\#v_0\#u_1\#v_1 \ldots$ y $\rho_I(p) = x$ entonces x es sólo el punto de intersección de todos los intervalos cerrados $[\overline{u_i}, \overline{v_i}]$. Si $p = u_0\#u_1\#\ldots$, y $\rho_C(p) = x$, entonces $(\overline{u_i})_{i\in\omega}$ es una sucesión de Cauchy de números racionales que convergen a x y en consecuencia $\mid \overline{u_k} - x \mid \leq 2^{-k}$ $\forall k \in \omega$. Por lo tanto, podemos asociar a $u_0\#u_1\#\ldots$ la sucesión $(I_n)_{n\in\omega}$ de intervalos anidados, donde $I_n := [\overline{u}_{n-2^{-n}}; \overline{u}_{n+2^{-n}}]$.

Para estudiar la computabilidad de los números reales, la representación de Cauchy es la más adecuada para definir la computabilidad en la recta real, pues tanto el conjunto de números computables como el de funciones computables solo coinciden bajo esta representación.

Definición 2.2.2. Un número $x \in \mathbb{R}$ es computable, si y solo si, se puede generar una sucesión infinita $p = (q_1, \varepsilon_1, q_2, \varepsilon_2, \ldots, q_n, \varepsilon_n \ldots)$ de números racionales tales que

$$\mid x - q_n \mid \leq \varepsilon_n \qquad \text{con} \qquad \lim_{n\to\infty} \varepsilon_n = 0$$

. Usando una codificación estándar de números racionales $\overline{p} = (q_1, \varepsilon_1, \ldots, q_n, \varepsilon_n, \ldots) \in \mathbb{Q}^{\mathbb{N}}$.

En adelante por simplicidad ρ_C será sólo ρ y diremos que p es un $\rho-$nombre para $x \in \mathbb{R}$.

Ejemplo 2.2.1.

- Todo número racional es computable: consideremos $r \in \mathbb{Q}$. Definamos $u \in \Sigma^*$ por $\overline{u} = r$, y $q := u\#u\#\ldots = (u\#)^\omega \in \Sigma^\omega$, entonces q es computable y $\rho(q) = r$.

- $\sqrt{2}$ es computable:
  Definamos $f : \omega \longrightarrow \omega$ por $f(n) = k \in \omega$ con $k^2 < 2 \cdot 2^n \leq (k+1)^2$, entonces f es computable. Sea $\overline{u}_n := f(n) \cdot 2^{-n}$, entonces $p = u_0\#u_1\#u_2 \ldots$ es computable y $\rho(p) = \sqrt{2}$. ♦

Teorema 2.2.1. ***(El límite de una sucesión computable es computable.)***
Sea $(y_i)_{i\in\omega}$ una succeción de números reales $(\nu_{\mathbb{N}}, \rho)-$computable tal que, para todo $i, j \geq m(n)$,

se tiene que $\mid y_i - y_j \mid < 2^{-n}$ *para alguna función computable* $m : \omega \to \omega$. *Entonces el real*

$$x := \lim_{i\to\infty} y_i$$

es computable

Demostración. Para algunos $i, j \in \omega$ una palabra $u_{ij} \in \mathbf{dom}(\nu_{\mathbb{Q}})$ puede ser computada de modo que

$$y_i = \rho(u_{i0}\#u_{i1}\#\ldots)$$

Sea $\nu_i := u_{m(i+1),(i+2)}$ para todo $i \in \omega$ entonces

$$q := \nu_0\#\nu_1\#\ldots \qquad \text{es computable.}$$

Para todo $k > i$ tenemos

$$\begin{aligned}\mid \overline{\nu}_i - \overline{\nu}_k \mid &\leq \mid \overline{u}_{m(i+1),i+2} - y_{m(i+1)} \mid + \mid y_{m(i+1)} - y_{m(k+1)} \mid + \mid y_{m(k+1)} - \overline{u}_{m(k+1),k+2} \mid \\ &\leq 2^{-i-2} + \max\left(2^{-i-1}, 2^{-k-1}\right) + 2^{-k-2} \\ &< 2^{-i}\end{aligned}$$

y

$$\begin{aligned}\mid \overline{\nu}_i - x \mid &\leq \mid \overline{u}_{m(i+1),i+2} - y_{m(i+1)} \mid + \mid y_{m(i+1)} - x \mid \\ &\leq 2^{-i-2} + 2^{-i-1} \\ &< 2^{-i}\end{aligned}$$

Luego $x = \rho(q)$ entonces x es $\rho-$computable. □

Ejemplo 2.2.2. Definamos:

$$x_i := \sum_{k=0}^{i} \frac{1}{k!} \qquad \text{entonces} \qquad e = \lim_{i\to\infty} x_i$$

Obviamente la sucesión $(x_i)_{i\in\mathbb{N}}$ es $(\nu_{\mathbb{N}}, \nu_{\mathbb{Q}})-$computable por lo tanto $(\nu_{\mathbb{N}}, \rho)-$computable. Definamos $e(n) := n + 1$, entonces $[x_i - x_j] \leq 2^{-n}$ para $i, j \geq e(n)$; luego por el teorema anterior obtenemos que el número e es computable. ♦

Así como existe una representación que hace a los números reales $\rho-$computables y en general computables, vamos a encontrar caracterizaciones para subconjuntos $X \subseteq \mathbb{R}$ que son $\rho-$abiertos, r.e. y recursivos. También podemos ver que conjuntos de $\mathbb{R}$ como $\{x \in \mathbb{R} \mid x > a\}$ para algún $a \in \mathbb{R}$ computable y conjuntos de $\mathbb{R}^2$ como $\{(x, y) \in \mathbb{R}^2 \mid x > y\}$ son recursivamente enumerables; estas y otras propiedades para los conjuntos de $\mathbb{R}$ y $\mathbb{R}^2$ pueden ser vistas a detalle en [2].

2.3. Computabilidad de funciones reales

Como vimos en la sección anterior los números reales y conjuntos de números reales son computables con una representación adecuada, de la misma manera muchas de las funciones estudiadas en el análisis clásico son computables.

Definición 2.3.1. Una función real $f :\subseteq \mathbb{R} \longrightarrow \mathbb{R}$ es computable (más precisamente $(\rho, \rho)-$computable) si y solo si existe alguna máquina Tipo 2 que al ingresar un $\rho-$nombre para algún $x \in \mathbb{R}$ genera un correspondiente $\rho-$nombre para $y := f(x)$.

En otras palabras, siempre que tengamos dos conjuntos X, Y junto con sus *representaciones*, es decir funciones parciales sobreyectivas

$$\delta :\subseteq \{0,1\}^{\mathbb{N}} \longrightarrow X \qquad \text{y}$$

$$\delta' :\subseteq \{0,1\}^{\mathbb{N}} \longrightarrow Y$$

podemos decir que

$$f : X \longrightarrow Y$$

es una función $(\delta, \delta')-$computable, siempre que exista una máquina de Turing que transfiera cada $\delta-$nombre de una entrada $x \in X$ a un $\delta'-$nombre del valor de la función $f(x) \in Y$. De la definición anterior y del teorema 2.1.1 podemos concluir que toda función real computable es continua; de forma más precisa, toda función real $(\rho^n, \rho)-$computable es continua.

Teorema 2.3.1. *Las siguientes funciones reales son computables:*

1. $(x_1, \ldots x_n) \longmapsto c$ *donde* $c \in \mathbb{R}$ *es computable.*

2. $(x_1, \ldots x_n) \longmapsto x_i$ *con* $1 \leq i \leq n$

3. $x \longmapsto -x$

4. $(x, y) \longmapsto x + y$

5. $(x, y) \longmapsto x \cdot y$

6. $x \longmapsto \dfrac{1}{x}$

7. $(x, y) \longmapsto \text{mín}(x, y), \qquad (x, y) \longmapsto \text{máx}(x, y)$

8. $x \longmapsto \mid x \mid$

9. *Toda función polinómica en n variables con coeficientes computables.*

10. $(i, x) \longmapsto x^i$ *para* $i \in \mathbb{N}$ y $x \in \mathbb{R}(0^0 := 1)$

Demostración. Se presentara la prueba de las propiedades (1), (2), (3) y (9); las pruebas de las otras propiedades se podrán ver a detalle en [3]

1. Sea M una máquina Tipo 2 con n cintas de entrada que en cada entrada calcula algún $\rho-$nombre de c. Entonces f_M realiza la función constante con valor c.

2. Sea M una máquina Tipo 2 con n cintas de entrada que copia la i-ésima cinta entrada a la cinta de salida. Entonces f_M realiza la i-ésima proyección.

3. Existe una función de palabras computable $f : \Sigma^* \longrightarrow \Sigma^*$ tal que $-\nu_{\mathbb{Q}}(\omega) = \nu_Q f(\omega)\ \forall \omega \in \mathbf{dom}(\nu_{\mathbb{Q}})$.

   Existe una máquina M Tipo 2 que transforma cada entrada $P := \#u_0\#u_1 \ldots \in \mathbf{dom}(\rho)$ (donde $u_i \in \mathbf{dom}(\nu_{\mathbb{Q}})$) en la sucesión $q := \#f(u_0)\#f(u_1)\ldots$; obviamente $\rho(q) = -x$ si $\rho(p) = x$, entonces, f_M realiza el negativo en $\mathbb{R}$.

9. Todo polinomio de grado 0 (esto es $f(x_1, \ldots, x_n) := c$, donde c es una constante computable) es computable por (1). Supongamos que todos los polinomios de grado k con coeficientes computables son computables. Si f_{k+1} es un polinomio de grado $k+1$ con coeficientes computables, entonces $f_{k+1}(x_1, \ldots, x_n) = f_k(x_1, \ldots, x_n) \cdot pr_i(x_1, \ldots, x_n)$

para algún polinomio de grado k con coeficientes computables y algún i.

Por inducción y propiedades 2 y 5 obtenemos que todo polinomio con coeficientes computables es computable.

□

De este teorema y el teorema 2.1.1 se pueden concluir que si $f, g :\subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$ son funciones computables y $a \in \mathbb{R}$ es un número computable, entonces funciones tales como:

$$x \longmapsto a \cdot f(x) \qquad x \longmapsto 1/f(x)$$
$$x \longmapsto f(x) + g(x) \qquad x \longmapsto f(x) \cdot g(x)$$
$$x \longmapsto \text{mín}(f(x), g(x)) \qquad x \longmapsto \text{máx}(f(x), g(x)$$

son también computables.

Otra forma de definir las funciones computables sin definir los argumentos de entrada y salida es la siguiente: una función $f : \mathbb{R} \longrightarrow \mathbb{R}$ es computable, si y sólo si, se puede generar una sucesión de coeficientes de polinomios racionales $p_n \in \mathbb{Q}[x]$ que se aproximan a f hasta aumentar la precisión de manera uniforme, en intervalos compactos que eventualmente agotan toda la línea; esta última definición se puede expresar de la siguiente forma:

$$(\forall n \in \mathbb{N})\ (\forall x \in [-n, n])\ (\forall m \geq n)\ \left(|f(x) - p_m(x)| \leq 2^{-m}\right) \tag{2.1}$$

Vamos a denotar por $C(X)$ el conjunto $\{f :\subseteq \mathbb{R} \longrightarrow \mathbb{R} \mid f \text{ es continua y } \mathbf{dom}(f) = X\}$ y su representación estará dada por $\delta_{\mathbb{R}} :\subseteq \Sigma^\omega \longrightarrow C(\mathbb{R})$ para $C(\mathbb{R})$, que estará definida de la siguiente manera:

$\delta_{\mathbb{R}}(p) = f$ si y solo si existen palabras $u_i, v_i, x_i, y_i \in \mathbf{dom}(\nu_{\mathbb{Q}})$ donde $i \in \omega$ con

$$p = u_0\#v_0\#x_0\#y_0\jmath u_1\#v_1\#x_1\#y_1\jmath \ldots$$

tal que para todos los números racionales a, b, c, d:

$$f[a, b] \subseteq (c, d) \Longleftrightarrow (\exists i)(a = \overline{u}_i, b = \overline{v}_i, c = \overline{x}_i, d = \overline{y}_i)$$

para todo $p \in \Sigma^\omega$ y $f \in C(\mathbb{R})$

Definición 2.3.2. Un $[\rho \to \rho]-$nombre para alguna función $f \in C(\mathbb{R})$ es una sucesión de polinomios que satisfacen la expresión 2.1 vista anteiormente. La representación inducida es denotada por $[\rho \to \rho] :\subseteq \{0,1\}^{\mathbb{N}} \to C(\mathbb{R})$.

La naturalidad de esta definición se da en que un $[\rho \longrightarrow \rho]-$nombre para f puede servir como un "programa" para alguna máquina universal de Tipo 2 para evaluar uniformemente f en argumentos arbitrarios x.

Todas las nociones y definiciones que vimos a lo largo de este capítulo de reales simples $x \in \mathbb{R}$ pueden ser extendidas a vectores $(x_1, ..., x_d) \in \mathbb{R}^d$ y se hace de forma directa por medio de sucesiones de números racionales $(q_{n,1}, \varepsilon_{n,1}, ..., q_{n,d}, \varepsilon_{n,d})_n$. De la misma forma, podemos definir un ρ^d-nombre para $\overrightarrow{x}$ como una cadena infinita $(\sigma_1\sigma_2...\sigma_n...)$ donde, para cada $i = 1, ..., d$, la subcadena $(\sigma_{(n-1)d+i})_n$ es un $\rho-$nombre para x_i. Similarmente, ρ^* hace referencia a la computabilidad para vectores de longitud variable $\overrightarrow{x} \in \mathbb{R}^*$ con $\mathbb{R}^* := \bigcup_{n=0}^{\infty} \mathbb{R}^n$, tales que $(x_1, ..., x_n) \mapsto n$ es efectiva o computable. Finalmente, la computabilidad en funciones de varias variables se puede definir de forma análoga a la definición de la computabilidad en funciones de una variable, combinándola con los conceptos anteriores, por ejemplo, podemos definir un $[\rho^d, [\rho^n \to \rho^m]]-$nombre para funciones $(\overrightarrow{x}, f) \in \mathbb{R}^d \times C(\mathbb{R}^n, \mathbb{R}^m)$.

Capítulo 3

Aplicaciones al Álgebra Lineal

En este capítulo presentaremos algunas aplicaciones de la Teoría de Efectividad Tipo 2, que explicamos en las secciones previas, en resultados del Álgebra Lineal. Nos enfocaremos en dos proposiciones que muestran funciones que relacionan las matrices con las transformaciones lineales y mostraremos que dichas funciones son computables desde el punto de vista de la computabilidad Tipo 2.

Como pudimos ver a lo largo del capítulo anterior, las funciones continuas en la computabilidad y específicamente en la teoría Tipo 2 son de gran importancia, pues dichas funciones, al ser continuas, resultan siendo computables. Es así como se pueden encontrar resultados propios de la computabilidad que funcionan tanto para funciones que son computables como para aquellas que son continuas.

La Teoría de Computabilidad está fundamentada en varios conceptos que son de gran importancia, uno de estos es el *Teorema de la Máquina de Turing Universal*, que, básicamente, afirma la existencia de una función universal que es computable, y la cual puede ser usada para calcular cualquier otra función computable. Otro concepto fundamental es el *Teorema de Parametrización* (también llamado Teorema S_m^n o Smn). Este teorema nos permite expresar funciones de varias variables mediante otras de un número inferior de variables, fijando, para ello, algunas de las variables originales; es decir, nos permite hacer reducciones.

Estos teoremas están definidos sobre funciones computables y pueden ser llevados a funciones continuas de la siguiente manera:

Teorema 3.0.1. *Las funciones*

$$\begin{aligned} C(\mathbb{R}^d) \times \mathbb{R}^d &\longrightarrow \mathbb{R} & C(\mathbb{R}^{n+m}) \times \mathbb{R}^n &\longrightarrow C(\mathbb{R}^m) \\ (f, x) &\longmapsto f(x) & (f, x) &\longmapsto f(x, \cdot) \end{aligned}$$

son ambas computables

Teorema 3.0.2. *Para cualquier función computable*

$$f : \mathbb{R}^{n+m} \longrightarrow \mathbb{R}$$

existe una función computable

$$F : \mathbb{R}^n \longrightarrow C(\mathbb{R}^m)$$

tal que

$$(\forall x \in \mathbb{R}^n)\ (\forall y \in \mathbb{R}^m)\ (f(x, y) = F(x)(y))\,.$$

Además de estos teoremas, se desprende otro resultado que también es de gran utilidad:

Teorema 3.0.3. *La función composición*

$$\begin{aligned} C(\mathbb{R}^n, \mathbb{R}^m) \times C(\mathbb{R}^k, \mathbb{R}^m) &\longrightarrow C(\mathbb{R}^k, \mathbb{R}^m) \\ (f, g) &\longmapsto f \circ g \end{aligned}$$

es computable.

Estos teoremas no serán demostrados ya que la teoría que tenemos es insuficiente y no está dentro de los objetivos del trabajo. Sin embargo, se presentan pues son resultados de gran importancia para lo que expondremos a continuación. Si se quiere investigar sobre su demostración, esta se puede encontrar en [3].

3.1. Funciones sobre matrices

Una de las herramientas más importantes que se estudian en el Álgebra Lineal es la correspondencia entre las matrices $A \in \mathbb{R}^{m\times n}$ y las transformaciones lineales $F \in Lin(\mathbb{R}^n, \mathbb{R}^m)$. Cuando tenemos bases fijas, cada matriz A induce una transformación F_A, y dada una transformación F, ella es de la forma $F = F_A$ para una única matriz A. Sobre la efectividad de esta correspondencia se observa a simple vista que las matrices van a ser consideradas como $\rho^{m\times n}-$nombres y $[\rho^n \to \rho^m]-$nombres para las transformaciones continuas F.

Proposición 3.1.1. *La función determinante $A \longrightarrow det(A)$ es $(\rho^{n\times m}, \rho)-$computable, además la inversion matricial*

$$GL(\mathbb{R}^n) \longrightarrow GL(\mathbb{R}^n)$$

$$A \longmapsto A^{-1}$$

es $(\rho^{n\times n}, \rho^{n\times n})-$computable, donde $GL(\mathbb{R}^n) = \{A \in \mathbb{R}^{n\times n} : det(A) \neq 0\}$ es el grupo lineal general, es decir el conjunto de las matrices cuadradas invertibles de orden n con coeficientes en $\mathbb{R}$.

Antes de probar este resultado veamos una proposición que será de gran ayuda.

Proposición 3.1.2. *Sean $\{u_1, \dots u_n\}$ y $\{v_1, \dots, v_m\}$ bases de $\mathbb{R}^n$ y $\mathbb{R}^m$ respectivamente. Supongamos además que son computables. Entonces la función*

$$\Phi : \mathbb{R}^{m\times n} \longrightarrow Lin(\mathbb{R}^n, \mathbb{R}^m)$$

$$A = (a_{ij}) \longmapsto F_A$$

con $F_A(u_j) := \sum_{i=1}^{m} a_{ij}v_i$ para todo $1 \leq j \leq n$, es $(\rho^{m\times n}, [\rho^n \to \rho^m])-$computable y su inversa es $([\rho^n \to \rho^m], \rho^{m\times n})-$computable.

Demostración. Consideremos primero en caso donde las bases para cada espacio son ortonormales. Sea $x \in \mathbb{R}^n$, para este x vamos a tener que la correspondencia

$$(A, x) \longmapsto \Phi(A)(x) = \left(\sum_{j=1}^{n} a_{ij}x_j\right)_{i=1,\dots,n} \in R^m$$

es computable, ya que la suma y la multiplicación como operaciones algebraicas son computables, luego la computablidad de la función Φ, que a una matriz A le asocia $\Phi(A)$, se obtiene aplicando el teorema 3.0.2.
Para probar que su inversa es computable, tomemos $L \in Lin(\mathbb{R}^n, \mathbb{R}^m)$, entonces

$$\Phi^{-1}(L) = (L(u_1), \ldots, L(u_n)) \in \mathbb{R}^{m \times n}$$

ya que v_i denota la base ortonormal para $\mathbb{R}^m$ por hipótesis.

Para el caso de bases computables en general, el resultado se obtiene por medio de la transformación $A \longmapsto T \cdot A \cdot S^{-1}$ a partir de la invertibilidad efectiva de las matrices regulares S, T de acuerdo con la proposición 3.1.1. □

De esta proposición se obtiene como corolario que, en particular, la composición de transformaciones lineales $F \circ G = \Phi(\phi^{-1}(F), \ldots, \Phi^{-1}(G))$ es computable a través de la multiplicación de matrices; dicho resultado se obtiene de forma trivial, en comparación a la demostración del teorema 3.0.3.

Demostración. ***(Proposición 3.1.1).***
Al ser un polinomio, la computablidad de la función determinante

$$det : \mathbb{R}^{n \times n} \longrightarrow \mathbb{R} \qquad\qquad A(a_{ij}) \longmapsto \sum_{\pi \in S_n} Sg(\pi) \cdot \prod_{i=1}^{n} a_{i,\pi(i)}$$

se desprende de la computabilidad de la suma y la multiplicación como operaciones algebraicas; aquí S_n hace referencia al grupo de permutaciones de $\{1, \ldots, n\}$.

Para mostrar la otra parte de la proposición, según la regla de Cramer los elementos de A^{-1} son cocientes de elementos de la traspuesta de la adjunta de A entre el determinante de A, el cual es distinto de 0 pues $A \in GL(\mathbb{R}^m)$, y ya que en estos términos la división entre números reales es computable, se obtiene que los elementos de A^{-1} son todos computables y de esto se concluye que la inversión de matrices es computable. □

En particular, una matriz regular es computable, si y solo si, su inversa también es computable; esto implica que la transformación de matrices bajo el cambio de bases (computables),

es computable. Este último resultado también es válido para transformaciones lineales con respecto a bases ortonormales de $\mathbb{R}^n$ y $\mathbb{R}^m$.

Al igual que los resultados previos, podemos analizar la computabilidad Tipo 2 de resultados matemáticos de diversas teorías, Topología, Análisis, Teoría de la Medida, etc. Nuestro objetivo era simplemente, exponer un pequeño resumen de los conceptos básicos de esta rama de la computabilidad para las personas interesadas en estos temas.

Conclusiones

- Los modelos de computación se pueden considerar como la clave para cualquier investigación en computabilidad teórica y permiten formalizar con elegancia nociones como "algoritmo" y "problema" en un marco matemático sólido.

- Alan Turing estableció los cimientos teóricos sobre los cuales está fundamentada la computabilidad, al ofrecer una noción para los conceptos de algoritmo y de computador universal.

- La Máquina de Turing a pesar de ser muy diferente a los computadores actuales, captura bastante bien sus principales capacidades de procesamiento y proporciona un marco para pruebas formales.

- Existen modelos de computación diferentes al modelo clásico, pero que basados en las máquinas de Turing, pueden simplificar y expresar de una mejor manera problemas en los que la computabilidad clásica presenta dificultades para su solución.

- La Teoría de Efectividad Tipo 2 extiende la teoría de computabilidad clásica y la conecta con el análisis abstracto. Es por esto que las máquinas Tipo 2 comparten la sintaxis de una máquina de Turing clásica (cintas, cabezales, configuración, etc), pero tienen una semántica modificada basada en contribuciones principalmente dadas por Grzegorczyk.

- La Teoría de Efectividad Tipo 2 estableció el análisis computable como una rama de la

teoría de la computabilidad, que estudia funciones en los números reales y conjuntos relacionados que pueden ser calculados por máquinas como las computadoras digitales.

- Hemos investigado la computabilidad de problemas de Álgebra Lineal: matrices y transformaciones lineales en espacios de vectores reales de dimensiones finitas, usando, aparte del modelo algebraico, un modelo fundamentado en el análisis computable que respeta inherentemente la estabilidad numérica.

Bibliografía

[1] Caicedo, X. (1990). *Elementos de lógica y calculabilidad.* Bogotá, Colombia: Una empresa docente.

[2] Weihrauch, K. (1995). *A Simple Introduction to Computable Analysis.* Hagen, Alemania: FernUniversität.

[3] Weihrauch, K. (2000). *Computable Analysis An Introduction.* Berlín, Alemania: Springer.

[4] Grzegorczyk, K. (1955). *Computable Functionals.* Varsovia, Polonia: Fundamenta mathematica, 42, 168 - 202.

[5] Ziegler, M. y Brattka, V. (2004).*Computability in linear algebra.* Berlín, Alemania: Theoretical Computer Science, 326, 187-211.

[6] Sicard Ramírez, A. (2012). *Máquina universal de turing: algunas indicaciones para su construcción.* Medellín, Colombia: Revista Universidad EAFIT, 33(108), 61-106.

[7] Sicard Ramírez, A. (2012). *Máquinas de Turing.* Medellín, Colombia: Revista Universidad EAFIT, 32(103), 29-45.

[8] Blum, L., Cucker, F., Shub, M. y Smale, S. (1998). *Complexity and real computation.* Nueva York, Estados Unidos: Springer.

[9] Schulz, H. (2000). *Type two theory of effectivity and real PCF.*, Berlín, Alemania: Theoretical Computer Science, 35, 191-203.

[10] Brattka, V. (2003). *Plottable real number functions and the computable graph theorem.* Múnich, Alemania: SIAM J. Comput., 38(1), 303–328.

[11] Church, A. (1936). *A Formulation of the Simple Theory of Types.* Hollister, Estados Unidos: The Journal of Symbolic Logic, 5(2), 56-68.

[12] Turing, A.M. (1937). *On Computable Numbers, with an Application to the Entscheidungsproblem.* Londres, Inglaterra: Proceedings of the London Mathematical Society, 2(42), 230-265.