UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

MAQUINAS DE TURING Y ARITMETICA RECURSIVA

T E S I S

Que para obtener el Título de

M A T E M A T I C O

P r e s e n t a

AGUSTIN GONZALEZ FLORES

México, D. F.

1989

INTRODUCCION

El objetivo principal de esta Tesis ha sido *desarrollar un Programa de aplicación para estudiantes de la carrera de Matemáticas en la Facultad de Ciencias de la UNAM*. En particular, para aquellos interesados en cursar la Materia *Introducción a Las Funciones Recursivas y Computabilidad*. Brindándose a éstos la oportunidad de poder utilizar el presente texto como libro de consulta para dicho curso.

En la parte Teórica se tiene como propósito establecer la siguiente relación entre La Aritmética Recursiva y Las Máquinas de Turing: *Una Función Aritmética es Recursiva si y sólo si es Turing-Calculable*. Para esto, en el capítulo I, se da una caracterización de la noción de Algoritmo, se define una cierta función g, cuyo dominio es el conjunto de palabras sobre un alfabeto A de n elementos, llamada *Numeración de Godel* y se establecen los argumentos necesarios para obtener la siguiente conclusión: *La caracterización de la noción de Algoritmo se reduce a la caracterización de las Funciones Aritméticas que son Calculables*.

En el capítulo II se definen los conceptos fundamentales de la *Teoría Constructiva*. Estos conceptos son el de Función Calculable y el de Relación Decidible. Además se establece la conección entre dichos conceptos. El capítulo termina probándose la invariancia del concepto de Decidibilidad Absoluta bajo la numeración de Gödel.

En el capítulo III, se plantea el concepto Matemático exacto de *Máquina de Turing* como reemplazo del concepto intuitivo de *Algoritmo*. Para esto se menciona la llamada Tesis de Church: *Toda Función Intuitivamente Calculable es Turing-Calculable*. Además se describe *el Método de Turing*, estableciéndose los requerimientos necesarios para definir el concepto de *Máquina de Turing*.

En el capítulo IV se define el concepto de *Máquina de Turing* y se establecen *Las Máquinas Elementales*. Además se definen los conceptos de *Configuración de una Máquina de Turing* y de *Equivalencia de Máquinas de Turing*, este último de gran utilidad en la elaboración del Programa y en la sección IV.5 . También se definen los conceptos de *Función Turing-Calculable* y de *Relación Turing-Decidible*. La sección IV.5 , llamada Combinación de Máquinas de Turing, muestra cómo construir máquinas complicadas combinando máquinas simples; el modo que se emplea para combinarlas, análogo al que se utiliza para construir los (diagramas de flujo) utilizados en la programación de computadoras electrónicas, es llamado *Diagrama* o *Digráfica finita*. Además se menciona como construir La Máquina de Turing representada por una Digráfica, lo cual es muy importamnte en el PROGRAMA. Para terminar este capítulo se definen, a partir de las Máquinas elementales, Máquinas que serán de gran utilidad en la sección VI.2 (en donde se demuestra La Turing-Calculabilidad de Las Funciones Recursivas) y también en el Programa.

En el capítulo V se define el concepto de *Función Aritmética Recursiva (R.P)*[1] y los conceptos de Predicado Recursivo (R.P) y Función característica de un Predicado. Además se mencionan algunas proposiciones que nos sirven para poder construir nuevas funciones recursivas y nuevos predicados recursivos. Se define el concepto de Función Recursiva Primitiva por Casos, el cual es de gran utilidad en las secciones VI.3 y VI.4 (en donde se demuestra La Recursividad de Las Funciones Turing-Calculables). Se definen el operador μ no acotado y el operador μ acotado; y se establece cómo definir una nueva Función Recursiva Primitiva aplicando el operador μ acotado a un Predicado Recursivo Primitivo. Para terminar se definen las Funciones σ que serán de gran utilidad en las secciones VI.3 y VI.4 .

(1) R.P. significa Recursivo(a) Primitivo(a).

En el capítulo VI se establece la relación (mencionada al principio) entre La Aritmética Recursiva y Las Máquinas de Turing, junto con su demostración. Es importante decir que tal demostración se encuentra en el libro *Enumerability, Decidability, Computability* de H. Hermés; pero debido a su importancia ha sido incluida en el presente texto para comprender con mayor claridad el trabajo que realiza el Programa.

Como complemento de la teoría, se han incluido ejemplos de los conceptos más importantes que aparecen; lo cual tiende a facilitar su comprensión. Además se incluyen tres apéndices. En el apéndice A se introduce el concepto de *Deducción*, el cual nos ayuda a remarcar la conexión entre Procedimientos Generales y Algoritmos.

En el apéndice B se demuestra el siguiente resultado: partiendo de la suposición de que conocemos una Máquina M' que realiza el cálculo del valor de una función f n-aria $(n\geq1)$ al ser aplicada sobre el primer cuadro vacío a la derecha de los argumentos, entonces podemos describir una Máquina M (con la ayuda de M') que calcula a f en el sentido de la definición de Turing-Calculabilidad. Esto es muy importante puesto que, en el *Programa, las Máquinas que llevan a cabo el cálculo del valor de una función n-aria $(n\geq1)$ serán definidas de modo que realicen su trabajo al ser aplicadas sobre el primer cuadro vacío a la derecha de los argumentos.* Lo cual no representa limitación alguna, ya que para encontrar las máquinas que realicen el trabajo en el sentido de la definición de Turing-Calculabilidad sólo se necesita aplicar el resultado antes mencionado. En este segundo apéndice también se mencionan algunos ejemplos de Funciones Turing-Calculables y un ejemplo de Relación Turing-Decidible.

Por otro lado, *el objetivo principal del Programa es Construir Máquinas de Turing que Calculan Funciones Recursivas*. Esto se realiza del modo siguiente: dada una Función Recursiva junto con la sucesión de funciones que la definen, se puede construir (según se establece en la sección VI.2) una digráfica que representa a la Máquina de Turing que calcula a la función en forma estandar; y una opción (del Programa) llamada *Ligador de Tablas de una Digráfica* construye dicha máquina del modo que se menciona en la seccion IV.5 .

Para aclarar todo lo que el Programa puede realizar y cómo, consúltese el apéndice C (el cual es un manual para poder utilizar dicho Programa).

AGRADECIMIENTOS:

Agradezco especialmente al Maestro Carlos Torres Alcaráz, de quien surgió la idea del presente trabajo, por haberme brindado la oportunidad de realizarlo y por haber aceptado la dirección del mismo. También quiero dar las gracias a los profesores M.F.C. Rafael Rojas Barbachano, M.F.C. José Alfredo Amor, Mat. María de la luz Nuñez Morales y Mat. José Chacón Castro; por sus valiosos comentarios con respecto al mismo. Finalmente gracias a Victor Hugo Dorantes González, estudiante de la Carrera de Matemático en la Facultad de Ciencias, por las asesorias brindadas con respecto a la elaboración del Programa.

UNAM, México D.F. Agustin González Flores

C O N T E N I D O

CAPITULO I

NOTAS INTRODUCTORIAS SOBRE LOS ALGORITMOS

El concepto intuitivo de *ALGORITMO* es algo conocido en prácticamente todas las ramas de las *Matemáticas*. En esta primera parte se intenta hacer de este concepto algo más preciso, lo cual es esencial para el presente trabajo.

I.1 Los Algoritmos vistos como Procedimientos Generales.

Pretendemos substituir el concepto intuitivo de *ALGORITMO* con el concepto más exacto de *Procedimiento General*, entendiéndose un Procedimiento General como el modo de obrar con los objetos de un género.

Si bien la forma específica como una teoría matemática se extiende tiene diversos aspectos, uno de ellos es característico de muchos desarrollos: suele suceder que en un principio la atención del matemático se centra en algunos hechos aislados de los problemas que le ocupan, pero pronto comienza a relacionar tales hechos entre sí. Surge con ello la voluntad de sistematizar la investigación para alcanzar por esa vía un dominio, que tiende a ser absoluto, del campo en cuestión. En ocasiones, tal sistematización consiste en separar los problemas en clases. De modo que cada una de ellas sea manejable por medio de Algoritmos.

De esta manera, en primera instancia, podemos considerar *UN ALGORITMO* como un procedimiento general que para cada pregunta específica de una clase proporciona la respuesta correcta, por medio de un simple cálculo, conforme a un método fijo.

Ejemplos de procedimientos generales se pueden exhibir en cualquier disciplina de las Matemáticas. Basta pensar en el procedimiento para la suma de dos números naturales escritos en notación decimal, en el método de Cramer para resolver sistemas de ecuaciones lineales o en la descomposición de un número natural en factores primos.

Ahora bien, un procedimiento general se puede entender como un proceso cuyas reglas de ejecución están claramente especifidadas hasta el más mínimo detalle. Esto significa, entre otras cosas, que ha de ser posible escribir las instrucciones para su ejecución en un texto de longitud finita. Además, significa que no hay lugar a dudas con respecto a qué reglas son aplicables en cada paso posible y que no hay lugar para la creatividad de quien lo ejecuta, debiéndose trabajar esclavizado a las instrucciones dadas, mismas que determinan cada cosa hasta el último detalle.

Los requerimientos para que un procedimiento general sea un *Algoritmo* son muy estrictos. Para entender esto, debe ser claro que los caminos y formas que en matemáticas son utilizados para la descripción de un procedimiento general son en ocaciones demasiado vagos para cubrir lo requerido para ser un *Algoritmo*. Esto lo podemos ver, por ejemplo, en la descripción usual de métodos para la solución de un sistema de ecuaciones lineales. Entre otras cosas, en esta descripción estamos dejando abierto, a criterio de quien lo ejecute, en qué parte del camino las multiplicaciones y adiciones necesarias serán ejecutadas. Sin embargo, es claro, para cualquier matemático, que en este caso y en casos de la misma especie las instrucciones pueden ser reemplazadas por una descripción completa, de modo que no exista alternativa con respecto a cuál será el siguiente paso.

Un rasgo importante de los procedimientos generales se pone de manifiesto en la clasificación siguiente. Nos referimos a la posibilidad de proseguir o no la aplicación de algunos de ellos.

CLASIFICACION DE LOS PROCEDIMIENTOS GENERALES.

$$\text{Procedimientos Generales}\left\{\begin{array}{l} \text{Multiopcionales o Cálculos}\left\{\begin{array}{l}\text{Concluyentes}\\ \text{Iterativos}^{1}\end{array}\right.\\ \\ \text{Coercitivos o Algoritmos}\left\{\begin{array}{l}\text{Concluyentes}\\ \text{Iterativos}\end{array}\right.\end{array}\right.$$

Descripción.

Procedimiento General Multiopcional : Aquel en el que no se especifica en qué orden se deben aplicar las reglas.

Algoritmo : Procedimiento General en el cual en cada paso se tiene especificado en forma unívoca lo que debe hacerse a continuación.

Algoritmo Concluyente : Aquel que después de un número finito de pasos llega a su fin.

Algoritmo Iterativo : Aquel que puede ser llevado tan lejos como se deseé, sin que llegue necesariamente a un término.

Ejemplos :

1.- El procedimiento para jugar ajedrez es un ejemplo de Procedimiento general multiopcional, ya que las reglas para jugarlo no se aplican en un orden forzoso.

(1) Véase el apéndice A para mayor información.

2.- *Algoritmo Iterativo* para determinar la raíz cuadrada de un número natural.

Para poder describir este algoritmo, de manera más clara, lo iremos aplicando a un número particular (digamos 286225) . Es decir, nuestra tarea es hallar

$$\sqrt{286225}$$

Primer paso. Se divide el número en períodos de dos cifras a partir del punto decimal hacia la izquierda

$$\sqrt{28,62,25.}$$

Segundo paso. Se extrae la raíz cuadrada (aproximada) del primer período de la izquierda (28) y de éste se resta el cuadrado de la cifra hallada (cifra que al cuadrado es menor o igual a dicho período)

```
 √28,62,25 | 5
 -25       |------
   3       |
```

Tercer paso. A la derecha del resto (3) se baja el siguiente período (62) y, por otro lado, se dobla la raíz encontrada

```
 √28,62,25 | 5
 -25       |------
   3 62    | 10   { Doble de la raíz }
```

Cuarto Paso. Para encontrar la siguiente cifra de la raíz, debemos hallar un dígito δ tal que, el número

$$(1\,0\,\underline{\delta}') \times (\underline{\delta})$$

sea el más aproximado (menor ó igual) a 362 . En este caso $\delta = 3$, puesto que $(1\,0\,\underline{3}) \times (\underline{3}) = 309$. Por tanto, la siguiente cifra de la raíz es $\delta = 3$, y restando { (103) x (3) } a 362, tenemos

```
√‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
  28,62,25  | 5 3
 -25        |--------
  ‾‾‾
   3 62     | 1 0
  -3 09     |
  ‾‾‾‾‾
     53     |
```

Quinto Paso. Repetimos el tercer paso . Es decir, a la derecha del resto (53) se baja el siguiente período (25) y, por otro lado, se dobla la raíz encontrada

```
√‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
  28,62,25  | 5 3
 -25        |--------
  ‾‾‾
   3 62     | 1 0
  -3 09     |
  ‾‾‾‾‾
     53 25  | 1 0 6    {Doble de la raíz}
            |--------
```

Sexto Paso. Para encontrar la siguiente cifra de la raíz, debemos hallar un dígito β tal que, el número

$$(1\ 0\ 6\ \beta) \times (\beta)$$

sea el más aproximado (menor o igual) a 5325. En este caso $\beta = 5$, puesto que $(1\ 0\ 6\ \underline{5}) \times \underline{5} = 5325$.

Por tanto, la siguiente cifra de la raíz es $\beta = 5$, y restando { (1065) × (5) } a 5325 , tenemos

```
 √28,62,25 | 5 3 5
 -25       |--------
 ---       |
   3 62    | 1 0
  -3 09    |
  -----    |
     53 25 | 1 0 5
    -53 25 |--------
    ------ |
         0 |
```

Hemos terminado el proceso de extraer raíz cuadrada a 286225, que ha resultado ser un número cuadrado (puesto que al terminar el proceso obtuvimos un residuo igual a cero) cuya raíz cuadrada exacta es 535. Cuando, al aplicar el proceso a un cierto número, el residuo es distinto de cero, se podrá continuar con el proceso bajando períodos de dos ceros y colocando el punto decimal a la raíz en el momento de bajar el primer período de ceros, de esta forma podemos aproximarnos al valor de la raíz tanto como queramos.

Este ejemplo pone en relieve un rasgo muy importante de los procedimientos generales: a saber, que en su ejecución no caben las explicaciones del " porqué " se procede así. En este caso las reglas que se siguen (como, por ejemplo, dividir el número en períodos de dos dígitos) se muestran un tanto arbitrarias y sin lógica alguna. En fin de cuentas, la explicación del " porqué " un algoritmo resuelve una cierta clase de problemas es algo externa al algoritmo mismo.

I.2 Caracterización de Los Algoritmos.

Sin entrar en detalles diremos que todo procedimiento general opera con objetos concretos. La separación de estos objetos de cualesquiera otros es bastante clara. Estos objetos pueden ser, por ejemplo, piedrecillas como las usadas en los ábacos, símbolos como los usuales en matemáticas (es decir : ∂, Σ, π) o impulsos eléctricos como los utilizados actualmente por las máquinas computadoras. En este sentido, una operación consiste en *Transformar* configuraciones espaciales y/o temporales en otras configuraciones.

En la práctica es absolutamente esencial saber a qué material se aplica un procedimiento. Sin embargo, nuestro interés es tratar a los Algoritmos desde un punto de vista teórico. En este sentido, el material es *Irrelevante*. Si un procedimiento trabaja con ciertos materiales, entonces dicho procedimiento puede *transferirse* a todo material que los substituya.

Veamos los siguientes ejemplos:

1.- La suma de dos números naturales puede interpretarse como una *transformación*. Así, la siguiente tabla indica en que configuración se *transforma* cada pareja de configuraciones:

(1 , 5)	6
(39 , 12)	51
(14 , 63)	77

2.- *Transferencia del Procedimiento a otro Material.* Cada número natural (originalmente escrito en notación decimal) se puede traducir en una sucesión de trazos:

$$
\begin{array}{cccccc}
1 & 2 & 3 & . & . & . \\
\Updownarrow & \Updownarrow & \Updownarrow & & & \\
/ & // & /// & . & . & .
\end{array}
$$

Así, el procedimiento de (Sumar números naturales) se *tranfiere* al nuevo material como (Adhesión de trazos a una línea de trazos) :

$$
\begin{tikzcd}
(\ 2\ ,\ 3\ ) \arrow[rr, "+"] \arrow[d, "\text{Substituciones}"'] & & (\ 5\ ) \arrow[d] \\
(\ //\ ,\ ///\ ) \arrow[rr, "+'"] & & (\ /////\ )
\end{tikzcd}
$$

{ (+') indica que el procedimiento (+) se ha transferido }.

Para los efectos que aquí se persiguen, es conveniente considerar aquellos Algoritmos que se efectuan alterando Líneas de Signos (es decir, los objetos concretos son signos). Esta consideración se fundamenta en lo siguiente: De un Procedimiento General solo interesan aquellas cuestiones que son independientes del material al cual se aplica dicho Procedimiento. Cabe considerar, en tal caso, que *el Procedimiento se ha Transferido a Objetos Simbólicos*[1].

Por tanto, se considera suficiente el ocuparse tan sólo de los Algoritmos que se llevan a cabo por medio de la transformación de líneas de signos. De esta forma, *Un Algoritmo se aplica a expresiones (Palabras) de un Alfabeto específico.*

Definiciones.

Alfabeto .- Conjunto no vacío y finito de signos (o letras).

Palabra.- Línea finita de signos del Alfabeto.

Palabra vacía sobre un Alfabeto A .- Palabra que no contiene letras. Se le denota así : $\square_A$.

Palabra sobre un Alfabeto A .- Palabra *p* formada solamente con letras de A . Al conjunto de palabras sobre un alfabeto A se le denota P_A.

(1) Un procedimiento general no se aplica a objetos abstractos, sino a objetos concretos. Ello significa que al ejecutarlo se está ante objetos específicos, no ante las entidades representadas por ellos. Así, el algoritmo para la suma que todos conocen actúa sobre objetos tales como las expresiones "2", "3" o "450". MAS NO SOBRE NUMEROS NATURALES. También podemos mencionar el hecho de que el sistema decimal es un ábaco representado "simbólicamente", es decir, el Procedimiento del ábaco se ha transferido a objetos simbólicos.

Las letras de un alfabeto A que es utilizado para desarrollar un Algoritmo son *Irrelevantes*, pudiendo ser reemplazadas por otras cualesquiera de cualquier otro alfabeto B. Veamos la siguiente proposición.

Proposición :

Si $f: R \longrightarrow S$ (donde $R \subseteq P_A$ y $S \subseteq P_B$) es una biyección, tal que $f(\square_A)=\square_B$, entonces un Algoritmo X aplicable a palabras de R se transfiere en un Algoritmo Y aplicable a palabras de S de la siguiente manera:

Dada una sucesión $\bar{v} = (v_1, v_2, \dots, v_n) \in S^n$, para determinar el resultado de aplicar el Algoritmo Y a $\bar{v}$, hacemos lo siguiente: en primer lugar, puesto que f es una biyección, existe una única $\bar{u}=(u_1, u_2, \dots, u_n) \in R^n$ tal que $(f^{-1}(v_1), \dots, f^{-1}(v_n))= (u_1, \dots u_n)$. A continuación, a esta sucesión $\bar{u}$ le aplicamos el Algoritmo X para obtener $X(\bar{u})$, el cual es un subconjunto de R. Para terminar, aplicamos f a cada elemento de $X(\bar{u})$ para obtener precisamente la definición de $Y(\bar{v})$. Es decir, $Y(\bar{v}) = f(X(\bar{u}))$.

Veamos el siguiente Diagrama:

$$\begin{tikzcd}
\bar{u} \arrow[r, shift left, "f"] \arrow[d, "X"'] & \bar{v} \arrow[l, shift left, "f^{-1}"] \arrow[d, "Y"] \\
X(\bar{u}) \arrow[r, "f"] & f(X(\bar{u})) = Y(\bar{v})
\end{tikzcd}$$

Nota : En esta demostración, al escribir f aplicada a un cierto conjunto de palabras C, se está representando el conjunto de palabras que se obtiene aplicando f a cada elemento de C.

Ejemplo:

Sean $A = \{0,1,2,3,4,5,6,7,8,9\}$ y $B = \{ / \}$.

Y sea $f : \mathbb{N} - \{0\} \longrightarrow \{ /, //, ///, \ldots \}$ tal que

$$f(\square_A) = \square_B \text{ , y}$$

$$f(n) = /\ldots/ \text{ (n-trazos) } \quad \forall\, n \in \text{Dom. de } f.$$

Claramente, f es biyectiva. Por lo tanto, el Algoritmo X para la suma de números naturales se transfiere en el Algoritmo Y de la adhesión de trazos a una línea de trazos, de la siguiente manera:

Dado, por ejemplo, $\bar{v} = (/,//)$ entonces, según lo expuesto, $\bar{u} = (1,2)$ y, $X(\bar{u})$ significa aplicar el Algoritmo de la suma a 1 y 2. Por lo tanto, $X(\bar{u})=3$. En consecuencia,

$$Y(\bar{v}) = f(X(\bar{u})) = f(3) = ///$$

que es precisamente lo que queríamos demostrar. Veamos el diagrama correspondiente:

$$\begin{tikzcd}
\{1,2\} \arrow[r, shift left, "f"] \arrow[d, "X"'] & \{/,//\} \arrow[l, shift left, "f^{-1}"] \arrow[d, "Y"] \\
\{3\} \arrow[r, "f"] & \{///\}
\end{tikzcd}$$

Como acabamos de ver, es suficiente considerar sólo Algoritmos que operan con palabras de algún alfabeto. Además, como las letras y las palabras de cualquier alfabeto son irrelevantes y pueden ser reemplazadas por otras cualesquiera, en particular, es posible reemplazarlas por *NUMEROS NATURALES*. Es decir, cabe la posibilidad de que el alfabeto sea $A = \{ / \}$ y $P_A = \mathbb{N}$.

Conclusión : *Es suficiente considerar ALGORITMOS aplicables a números naturales.*

Además, como a continuación se indica, todo conjunto de palabras P_A es sumergible en $\mathbb{N}$ de un modo efectivo. Así, *todo Algoritmo es transferible a números naturales.*

I.3 Numeración de Gödel (También llamada ARITMETIZACION).

En principio, el alfabeto A puede consistir en una sola letra, a saber $A = \{ / \}$. Las palabras de este alfabeto, aparte de la palabra vacía, son : /, //, ///,..., etc. Estas palabras pueden ser identificadas con los números naturales : 0, 1, 2,..., etc. Tal proceso es llamado *estandarización* del material.

Considerar un Alfabeto de una sola letra no constituye una limitación, veamos:

Si A es un alfabeto formado por n letras (Es decir, $A = \{ a_1, a_2, \ldots, a_n \}$) y P_A es el conjunto de palabras sobre A, entonces podemos asociar las palabras de P_A con *Números Naturales* (Esto es, palabras de un alfabeto que esta formado por un solo elemento) por medio de una función que llamaremos g. La representación g de P_A es llamada *Numeración de Gödel* y $g(p)$ es el *Número de Gödel* de la palabra p.

Los siguientes son los requerimientos para tal Aritmetización:

Se debe especificar una función $g : P_A \longrightarrow \mathbb{N}$ con las características siguientes

(*i*) g es inyectiva.

(*ii*) hay un Algoritmo que permite *Calcular Efectivamente* y en un número finito de pasos al número $g(\rho)$ para cada $\rho \in P_A$.

(*iii*) para cada $x \in \mathbb{N}$, es posible *Decidir Efectivamente* si $x \in g(P_A)$ o $x \in g(P_A)$. Es decir, podemos saber si x es número de Gödel de alguna palabra de P_A.

(*iv*) hay un algoritmo mediante el cual es posible reconstruir la palabra $g^{-1}(x)$ para cada $x \in g(P_A)$. Tal reconstrucción es finita.

Definición de la función g.

Tómese la correspondencia

$$\begin{array}{cccccc} a_1 & a_2 & . & . & . & a_n \\ \Updownarrow & \Updownarrow & . & . & . & \Updownarrow \\ 1 & 2 & . & . & . & n \end{array}$$

Se define $g(\square_A) = 0$, y

$g(a_{i1}, \ldots, a_{im}) = i1 \ . \ . \ . \ im$, que es un número natural escrito en base $(n+1)$. Puesto que $1 \leq ij \leq n$ para cada $1 \leq j \leq m$. Esta definición satisface las condiciones (*i*), (*ii*), (*iii*) y (*iv*).

Ejemplo :

Sea $A = \{ a_1, a_2, \ldots, a_9, a_h, a_k \}$ y sea $\rho = a_1\ a_9\ a_k\ a_2$, entonces $g(\rho) = 1\ 9\ k\ 2_{12} = 3\ 1\ 5\ 8_{10}$, puesto que

$$\begin{aligned} 2 \times 12^0 &= 2 \\ 11 \times 12^1 &= 132 \\ 9 \times 12^2 &= 1296 \\ 1 \times 12^3 &= 1728 \end{aligned}$$

y $\quad 1728 + 1296 + 132 + 2 = 3158$.

Sean $r = 1\ 5\ h\ 0\ k_{12}$ y $s = 1\ 5\ k\ h\ 0_{12}$. Es claro, que $r, s \notin g(P_A)$ ya que contienen al dígito cero en su escritura en base 12 y, por definición de g, cero no puede aparecer en ningún número de Gödel asociado a una palabra distinta de la vacía.

Para concluir el ejemplo, sea $x = 1\ 5\ 7\ 1\ 4_{10}$. Debemos reconstruir la palabra de la cual x es número de Gödel. Esto lo hacemos pasando x a base 12. De esta manera, tenemos:

$$\begin{aligned} 1\ 5\ 7\ 1\ 4 &= (\ 1\ 3\ 0\ 9 \times 12\ ) + 6 \\ 1\ 3\ 0\ 9 &= (\ 1\ 0\ 9 \times 12\ ) + 1 \\ 1\ 0\ 9 &= (\ 9 \times 12\ ) + 1 \end{aligned}$$

Por lo tanto,

$$\begin{aligned} 1\ 5\ 7\ 1\ 4 &= (((9 \times 12) + 1)\ 12 + 1)\ 12 + 6 \\ &= (\ 9 \times 12^3\ ) + (\ 1 \times 12^2\ ) + (\ 1 \times 12^1\ ) + (\ 6 \times 12^0) \\ &= 9\ 1\ 1\ 6_{12} . \end{aligned}$$

Con lo que $\quad g^{-1}(\ 1\ 5\ 7\ 1\ 4_{10}\ ) = a_9\ a_1\ a_1\ a_6$.

Comentario : En esta sección han aparecido de un modo natural las nociones de Calculabilidad y Decidibilidad. No obstante, la caracterización de lo que es un Algoritmo no se presupone a sí misma. Si bien el pasaje a los números naturales es indispensable para ahí caracterizar la noción, al hacerlo se está exhibiendo una función que a todas luces es Calculable, y un conjunto que es Decidible no porque concuerde con cierta definición sino porque es un hecho evidente.

Por lo tanto, todo Algoritmo sobre P_A es transformable en un Algoritmo aplicable a Números Naturales. Este Algoritmo es aplicable a todo $\mathbb{N}$, tomando en cuenta que, al aplicarlo a alguna $n \in \mathbb{N}$, se tendrá que $n \in g(P_A)$ o $n \notin g(P_A)$.

En el caso de que $n \in g(P_A)$, sucede lo siguiente: Como g es una biyección entre los conjuntos P_A y $g(P_A) \subset \mathbb{N}$, entonces cualquier Algoritmo aplicable a palabras de P_A se transfiere en un Algoritmo aplicable a palabras de $g(P_A)$, de un modo efectivo, según lo expuesto en la sección anterior. Ahora bien, en el caso de que $n \notin g(P_A)$, sucederá que, al aplicarle el Algoritmo transferido, simplemente no se obtendrá resultado alguno, ya que no hay forma de regresar de $n \in \{ \mathbb{N} - g(P_A) \}$ a alguna $p \in P_A$.

Definición (Función Aritmética).

Una función es llamada *Aritmética* si sus argumentos y valores son Números Naturales.

Definición (Función Calculable).

Una función es llamada *Calculable* si existe un Algoritmo concluyente (o terminante) que proporciona, para cada valor de su argumento, el valor de la función.

Conclusión :

La caracterización de la noción de *Algoritmo* se reduce, según lo expuesto, a la caracterización de las *Funciones Aritméticas* que son *Calculables*. En efecto :

1) Un Algoritmo transforma líneas de signos.
2) Las líneas de signos son palabras sobre un Alfabeto.
3) Las palabras sobre cualquier alfabeto se pueden reemplazar unívocamente por Números Naturales (Es decir, palabras de un alfabeto específico).
4) Todo Algoritmo es transferible a Números Naturales.

Por lo tanto, concentramos nuestra atención en la clase de las Funciones Aritméticas.

Comentarios :

1.- La palabra vacía puede o no convenir considerarla en ciertas reflexiones. Sin embargo, nada se pierde con la exclusión de la misma. De hecho, las palabras sobre un Alfabeto $A = \{ a_1, a_2, \ldots a_n \}$ se pueden mapear sobre las palabras no vacías de A usando la siguiente función J :

$J(\Box) = a_1$,

$J(w) = a_1 w$, si la palabra w contiene sólo símbolos que coinciden con a_1 (es decir : $w = a_1$, $w = a_1 a_1$, $w = a_1 a_1 a_1, \ldots,$ etc); y

$J(w) = w$, en los demás casos.

Si tal procedimiento se aplica a los Números Naturales representados por las palabras no vacías sobre el Alfabeto { / }, resulta que el número n queda representado por (n+1) trazos.

2.- Cabe esperar que algunos cálculos sean irrealizables físicamente, ya sea por limitaciones de tiempo o por ser contraria su realización con las leyes de la naturaleza. Por ejemplo, si quisieramos calcular el número $1000^{1000^{1000}}$, es posible que la existencia de un cálculo real sea contradictoria a las leyes de la naturaleza, ya sea porque no existe suficiente material en el universo para escribir el resultado en notación decimal o porque no hay un hombre que viva lo suficiente para poder escribir tal cálculo.

Por tanto, debemos extrapolar, es decir, asumir una situación ideal en la que no hay limitaciones de tiempo, espacio o material para desarrollar el procedimiento. Hablaremos, pues, de los Algoritmos posibles y no sólo de los realizables.

C A P I T U L O I I

CONCEPTOS FUNDAMENTALES DE LA TEORIA CONSTRUCTIVA

A partir de lo que hemos estudiado acerca del concepto de Algoritmo podemos derivar dos conceptos importantes: la Calculabilidad y la Decidibilidad. En este capítulo, se definirán estos conceptos estableciendo cierta relación entre ellos.

II.1 Funciones Calculables.

Definición (Calculabilidad).

Una función es Calculable si existe un Algoritmo terminante que proporciona, para cada valor de su argumento, el valor de la función.

Son calculables, por ejemplo, las funciones aritméticas $(X+Y)$, $(X \cdot Y)$, X^Y, en donde cualesquiera números naturales X e Y pueden ser tomados como argumentos de las funciones. Cuando decimos (de aquí en adelante) que los argumentos son números naturales, debemos darnos cuenta de que es más correcto, en este contexto, decir que los argumentos (si preferimos la notación decimal) son palabras sobre el alfabeto { 0, 1, . . . , 9 }. Sólo aquellas funciones cuyos argumentos y valores son palabras sobre algún alfabeto específico entran al dominio de nuestras consideraciones. Esto limita las consideraciones a dominios a lo más numerables para los argumentos de las funciones.

Comentarios :

(i) Cuántas funciones hay de $\mathbb{N}$ en $\mathbb{N}$?

De acuerdo a la Teoría de conjuntos, un número no numerable : $\aleph_0^{\aleph_0}$.

(ii) Cuántas de ellas son Calculables ?

Sólo un número numerable : $\aleph_0$. La causa de ello es que el conjunto de textos de longitud finita (por ejemplo, en idioma español) es numerable . Por lo tanto, sólo hay un numero numerable de Algoritmos.

II.2 Conjuntos y Relaciones Decidibles.

Ahora trataremos con una expresión que a menudo es utilizada en relación a los Algoritmos. Iniciamos con algunos ejemplos :

1.- Es Decidible si un número natural es primo o no.

2.- Es Decidible si un sistema de ecuaciones lineales tiene o no solución.

3.- No es decidible si una fórmula del cálculo de predicados es universalmente válida o no.

Tratando de hallar una estructura común, en estos ejemplos , podemos ver que en cada uno de ellos se menciona a dos conjuntos (K_1 y K_2) que mantienen cierta relación . Así, en (1) K_1 es el conjunto de los números primos y K_2 es el conjunto de los números naturales, en (2) K_1 es el conjunto de los sistemas de ecuaciones lineales que tienen solución y K_2 es el conjunto de todos los sistemas de ecuaciones lineales , y en (3) K_1 es el conjunto de las fórmulas del cálculo de predicados que son universalmente válidas y K_2 es el conjunto de todas las fórmulas del cálculo de predicados. Obsérvese que K_1 siempre es subconjunto de K_2.

Definición (Decidibilidad).

Sean K_1 y K_2 conjuntos de palabras sobre un alfabeto A, y sea $K_1 \subset K_2$. Se dice que K_1 es *Decidible Respecto a* K_2 si existe un Algoritmo terminante mediante el cual, para cada palabra $\rho \in K_2$, se puede saber de un modo efectivo si ésta pertenece o no a K_1. Tal Algoritmo es llamado Procedimiento de Decisión. Si K_1 es decidible respecto a P_A, entonces se dice que K_1 es *Absolutamente Decidible*.

Ahora bien, una relación de n términos se puede pensar como un conjunto de n-adas. Por lo tanto, el concepto de decidibilidad puede aplicarse también a relaciones.

Cualquier conjunto finito (y en consecuencia cualquier relación finita) es Absolutamente Decidible . Un procedimiento de decisión consiste en escribir todos los elementos del conjunto en una lista e ir checando, para cualquier n-ada , si ésta aparece o no en la lista.

Proposición : Sea K un conjunto de palabras sobre un alfabeto A, es decir $K \subseteq P_A$. Asignamos a este conjunto una función X_k, llamada la función carasterística de K (cuyo dominio es el conjunto P_A), definida así :

$$X_k(\rho) = \begin{cases} 0 & \text{si } \rho \in K. \\ 1 & \text{si } \rho \notin K. \end{cases}$$

En tal caso, K es *Absolutamente Decidible* si y sólo si la función característica X_k es *Calculable*.

Demostración.

(i) Supongamos que K es absolutamente decidible. entonces podemos determinar, para cada $p \in P_A$, si $p \in K$ o $p \notin K$. Por lo tanto podemos obtener el valor de la función $X_k(p)$. De esta manera queda establecido el Algoritmo para calcular X_k.

(ii) Supongamos que X_k es calculable. Entonces calculando el valor de $X_k(p)$ podemos decidir si $p \in K$ o $p \notin K$.

III.3 Invariancia del concepto de Decidibilidad Absoluta bajo la Numeración de Gödel.

Sea G la numeración de Gödel de las palabras sobre un alfabeto A de n elementos. Sea H un conjunto de palabras sobre A. Asignemos a H el conjunto $\mathbb{H}$ de los números de Gödel correspondientes a los elementos de H. Entonces tenemos la siguiente

Proposición : H es Absolutamente Decidible si y sólo si $\mathbb{H}$ es Absolutamente Decidible.

Demostración.

(i) Sea H absolutamente decidible. Entonces, dado un número $n \in \mathbb{N}$, determinamos si éste es número de Gödel de alguna palabra sobre A. Si este no es el caso, entonces $n \notin \mathbb{H}$. Si por otro lado n es el número de Gödel de una palabra $w \in P_A$, checamos si w pertenece a H . Y por lo tanto podemos decir si $n \in \mathbb{H}$.

(ii) Sea $\mathbb{H}$ absolutamente decidible. Entonces, dada una palabra $p \in P_A$, calculamos el número $G(p)$ y checamos si $G(p)$ pertenece a $\mathbb{H}$. de esta manera podemos determinar si $p \in H$.

C A P I T U L O   I I I

EL CONCEPTO DE MAQUINA DE TURING COMO REEMPLAZO MATEMATICO DEL CONCEPTO DE ALGORITMO

Para nuestros fines es conveniente reemplazar el concepto intuitivo de Algoritmo por un concepto Matemático exacto. A saber, por el concepto Matemático de Máquina de Turing. El primero en sugerir la identificación del concepto intuitivamente dado de *Algoritmo* con un concepto definido con exactitud fue *A. Church* en 1936.

La llamada Tesis de Church :

Toda Función Intuitivamente Calculable es Turing-Calculable,

es admitida hoy en día por la mayor parte de Lógicos y Matemáticos.

III.1 Máquinas y Algoritmos.

Además del concepto de Máquina de Turing, discutido aquí, hay otras sugerencias como reemplazos del concepto de *Algoritmo*. Discutiremos otra de estas sugerencias (*Función Aritmética Recursiva*) más adelante. Escogimos las Máquinas de Turing, como nuestro punto de partida, puesto que este camino es el más natural y de más fácil acceso.

Quedará claro, una vez definidas, que cualquier *Máquina de Turing* describe un *Algoritmo* . La única cuestión es, si cualquier Algoritmo puede ser llevado a cabo por una Máquina de Turing adecuada. La afirmación de que *Todos los Algoritmos se pueden abarcar por Máquinas de Turing* se acepta en el mismo sentido que lo es La Tesis de Church, la cual mencionamos en la página anterior.

Un hecho muy importante, del cual probaremos un caso más adelante, es que los conceptos matemáticos exactos sugeridos como reemplazos del concepto de Algoritmo son equivalentes.

Como ya establecimos, en el capítulo 1, un Procedimiento General esta descrito hasta el más mínimo detalle, así que no se necesita imaginación para llevarlo a cabo. Ahora bien, si todo esta determinado en detalle, entonces es posible aplicar el método por medio de una máquina.

Las maquinas pueden tener estructuras muy complicadas. La estructura de las máquinas es algo que pasaremos por alto. De hecho, no nos interesa saber cómo lo hacen sino qué es lo que hacen. En particular, nos ocuparemos de un tipo de máquina que es Matemáticamente fácil de tratar. Estas son *Las Máquinas de Turing*, de las cuales podemos esperar que *todo Algoritmo pueda ser ejecutado (después de ciertos ajustes) por una Máquina adecuada de este tipo*. En seguida se describe el Método de Turing analizando el comportamiento de un Calculista, un ser humano, que realiza las operaciones de acuerdo a dicho método.

III.2 Descripción del Método de Turing.

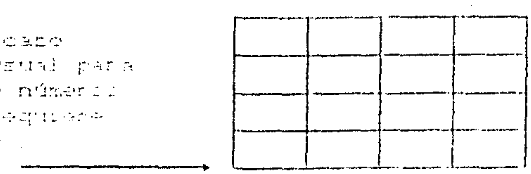
En esta sección establecemos los requerimientos necesarios para definir el concepto de *Máquina de Turing*. Para esto, mencionamos primero cuál es el *Material* utilizado por una Máquina de Turing, en seguida establecemos cuáles son *los pasos elementales* que se llevan a cabo mediante una Máquina de este tipo y, como parte final, se menciona la importancia de *los estados* de la Máquina.

El Material Utilizado.

Para fijar ideas, partimos de la suposición de que la tarea de un Calculista es determinar el valor de una función para un argumento dado, de acuerdo a las instrucciones dadas, las cuales contienen todos los detalles. El Calculista usa para sus cálculos una o varias hojas. Asumimos que tales hojas están divididas en cuadros. El calculista nunca escribirá más de un símbolo en un cuadro y sólo considera símbolos pertenecientes a un alfabeto finito $A = \{ a_1, \ldots, a_n \}$. El Argumento se escribe en la hoja, con estos símbolos, en el inicio del Cálculo.

Para algunos algoritmos es conveniente, sin duda, tener a disposición una superficie bidimensional para realizar los cálculos. Pensemos, por ejemplo, en el Algoritmo usual para la división de Números Naturales. Veamos :

Para llevar a cabo
el Algoritmo usual para
la división de números
naturales se requiere
una superficie
bidimensional



No obstante, la bidimensionalidad de la superficie no es algo indispensable. Las Máquinas de Turing utilizan una *CINTA unidimensional* que esta dividida en una secuencia lineal de cuadros. En el transcurso de los cálculos, suficientes cuadros estan disponibles en la *CINTA* (la cual se extiende infinitamente en ambas direcciones). Así, el aspecto de la cinta es semejante a esto :

. . .						. . .

Asumimos que la cinta esta provista con una dirección, en este sentido hablaremos de izquierda y derecha. Los cuadros de la Cinta estan vacíos con excepción de un número finito de ellos. Es conveniente incluir junto a los símbolos $a_1, \ldots, a_n$ al símbolo vacío como auxiliar. Lo representaremos por a_0 o $*$. Así, un cuadro esta vacío si y sólo si uno de los símbolos a_0 o $*$ esta escrito en él.

A la inscripción sobre la cinta se le llama *expresión de cinta*. Así, por ejemplo, si $A = \{a_1, a_2, \ldots, a_6\}$, son expresiones de cinta :

. . .	$*$	a_1	$*$	a_3	a_6	$*$	a_4	$*$	. . .

. . .	$*$	a_4	a_5	a_6	a_3	$*$	. . .

. . .	$*$	a_2	$*$	$*$	$*$	a_1	$*$	. . .

Donde los puntos suspensivos representan una secuencia infinita de cuadros vacíos.

Comentario : Al usar el Programa , un cuadro se considerará vacío si no tiene ningún símbolo escrito en él. Esto es por razones prácticas.

Los Pasos Elementales.

Fijamos nuestra atención en el proceso de cálculo según el método de Turing. Los cálculos se realizan de acuerdo a un conjunto de instrucciones finito. Intentaremos llevar a cabo tales instrucciones en forma *ESTANDARIZADA*. Lo primero de todo es aclarar que cualquier cálculo lo podemos pensar como un proceso que puede ser dividido en pasos que llamaremos *ELEMENTALES*.

Comentario: Tales pasos elementales podrían consistir, por ejemplo, en escribir una letra en un cierto cuadro vacío. La escritura en realidad es un proceso continuo. Sin embargo, debido a que procuraremos dividir todo proceso de cálculo en pasos elementales, es justificado para nosotros hablar de procesos discretos.

Un paso del cálculo nos lleva de una configuración a otra, la cual pasa a ser la base para el próximo paso. El proceso sólo se detiene cuando una cierta configuración es alcanzada.

Qué puede suceder en un *Paso Elemental del Cálculo* ?

En primer lugar, existe la posibilidad de que la expresión de cinta sea cambiada. En consecuencia, uno de los Pasos Elementales será aquel en el que un símbolo arbitrario a_j, que esté escrito en un cuadro, se cambio por un símbolo a_k $(j,k = 0,1,\ldots n)$. En particular, k puede ser igual a j, en cuyo caso la letra a_j no varía. De modo que cualquier cambio de una expresión de cinta consistirá en un cambio en un número finito de cuadros, y puede así ser dividido en pasos elementales de este tipo.

Es necesario mencionar que en cada instante el calculista pone especial atención a un cierto cuadro de la cinta. Llamamos a éste el *CUADRO EN INSPECCION* [1]. En general es necesario cambiar el cuadro en inspección en el transcurso del cálculo. El cambio de un cuadro en inspección a otro, que esta situado a una cierta distancia de él, podemos llevarlo a cabo a través de otro tipo de pasos elementales. Los cuales consisten en cambiar el cuadro en inspección por el cuadro que esta situado inmediatamente a su derecha o a su izquierda. El último de los pasos elementales es aquél que sirve para detener el proceso una vez que una cierta configuración se ha alcanzado.

En resumen, *los Pasos Elementales son* :

a_k : La impresión de un símbolo a_k ($k = 0,\ldots,n$) en el cuadro en inspección.

D : El movimiento, sobre la cinta, un cuadro a la derecha.

I : El movimiento, sobre la cinta, un cuadro a la izquierda.

P : El Detenerse (Parar).

Nos referiremos a estos pasos usando los respectivos símbolos que se encuentran en el lado izquierdo.

Importancia de los Estados para definir una Máquina de Turing.

Como acabamos de mencionar, en cada instante del proceso de operación de la máquina se pone especial atención a un cierto cuadro de la cinta. Ahora bien, en este cuadro en inspección esta impreso un símbolo perteneciente al conjunto $\{a_0,a_1,\ldots,a_n\}$ y para poder determinar cuál será el paso que llevaremos a cabo en un determinado instante necesitamos la ayuda de lo que llamaremos *ESTADOS*.

(1) En el programa, al cuadro en inspección lo llamaremos VENTANA.

Para entender esto vamos a suponer, por ejemplo, que queremos construir una Máquina de Turing cuyo trabajo al ponerla en operación, sobre cualquier cuadro de cualquier expresión de cinta, sea moverse un cuadro a la Derecha y Detenerse. Entonces lo que debe suceder es que al poner en marcha el proceso de operación de la máquina sobre cualquier cuadro de la cinta, estando en este cuadro cualquiera de los símbolos $a_0, a_1, \ldots, a_n$, la instrucción será D ; y una vez recorrida la máquina un cuadro a la derecha, sin importar qué símbolo se encuentre escrito en el nuevo cuadro, la instrucción será P. Por lo cual, en este caso, necesitamos un primer estado, que servirá para indicar cuándo se debe llevar a cabo el paso elemental D, y un segundo estado, que servirá para indicar cuándo se debe aplicar el paso elemental P.

Comentario : Algo semejante ocurre con un Calculista humano. Por ejemplo, ante la pareja de cifras (3,5) el calculista procede de un modo al sumarlos y de un modo distinto si de multiplicarlos se trata. Podemos decir, que en cada caso 'guarda' un 'estado' distinto.

Como veremos, nos podemos auxiliar de una tabla para describir el comportamiento de una Máquina de Turing. En el ejemplo que hemos mencionado la tabla esta formada según se muestra en la página siguiente :

Estados de la máquina	Símbolo que se encuentra en el cuadro inspeccionado	Paso a ejecutar	Nuevo estado a considerar
0	a_0	D	1
⋮	⋮	⋮	⋮
0	a_n	D	1
1	a_0	P	1
⋮	⋮	⋮	⋮
1	a_n	P	1

Más adelante demostraremos que los estados se pueden enumerar por cualesquiera Números Naturales. En este ejemplo enumeramos los estados con 0 y 1. Tomando en consideración que al poner en marcha el proceso de operación de la máquina ésta se encuentre en el estado cero, la tabla anterior describe a una máquina que se mueve un cuadro a la derecha y ahí se detiene.

En realidad, la tabla es una forma sintética de describir el comportamiento de la Máquina. En otras palabras, la tabla es la escritura de un Algoritmo.

C A P I T U L O I V

MAQUINAS DE TURING

En la definición de Máquina de Turing hablaremos de una matriz que consta de cuatro columnas. Por consiguiente, cada renglón de la matriz consta de un cuarteto de símbolos. El primero de éstos representa al estado (de la máquina) que debemos consultar para continuar el proceso; el segundo representa al símbolo escrito en el cuadro en inspección (en un determinado instante); el tercero es el paso elemental (o ACTO) a realizar (en ese mismo instante) y el cuarto símbolo representa el nuevo estado al que pasa la máquina. De esta manera, estamos capacitados para dar la definición de Máquina de Turing.

Sea $A = \{ a_1, \dots, a_n \}$, con $n \geq 1$, un Alfabeto dado. Sea $V = \{ a_0 \}$, donde a_0 representa al símbolo vacío. Sea $S = \{ I, D, P \}$ el conjunto de los pasos elementales I, D, P. Finalmente, sea $E_m = \{ 0, 1, \dots, m-1 \} \subset \mathbb{N}$, con $m \geq 1$, un segmento inicial del conjunto $\mathbb{N}$ de Números Naturales. A los elementos de E_m los llamaremos *ESTADOS* de la máquina.

IV.1 Definición.

Definición (de Máquina de Turing).

Una Máquina de Turing M sobre A es una matriz (tabla) de orden $m(n+1) \times 4$, definida de la forma siguiente:

Estado actual de la máquina	Símbolo que se encuentra en el cuadro inspeccionado	Acto a realizar	Nuevo Estado
0	a_0	b_0	c_0
0	a_1	b_1	c_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	a_n	b_n	c_n
1	a_0	b_{n+1}	c_{n+1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	a_n	b_{2n+1}	c_{2n+1}
2	a_0	b_{2n+2}	c_{2n+2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
m-1	a_0	$b_{(m-1)n+(m-1)}$	$c_{(m-1)n+(m-1)}$
m-1	a_1	$b_{(m-1)n+m}$	$c_{(m-1)n+m}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
m-1	a_n	$b_{mn+(m-1)}$	$c_{mn+(m-1)}$

Donde $b_i \in A \cup V \cup S$ y $c_i \in E_m$ para cada i tal que $0 \leq i \leq mn + (m-1)$.

Es claro que podemos identificar una Máquina de Turing por su tabla, ya que para cada pareja ja_i (con $j \in E_m$ y $a_i \in A \cup V$) existe exactamente una línea en M que empieza con ja_i. Además, dado un estado (j) y el símbolo a_i existente en el cuadro en inspección, quedan determinados (por la tabla) el acto y el nuevo estado al que hay que referirse para continuar con el proceso hasta hallar un acto de parada.

Cero es llamado *el estado inicial de* M. También denotaremos al estado inicial de M por C_m. C_m es el primer estado mencionado en la tabla que define a M. Vamos a considerar que al poner en marcha el proceso de operación de la máquina, ésta siempre se encuentra en el estado C_m. Además, es importante mencionar que los estados de una máquina pueden estar numerados (según se verá más adelante) por cualesquiera números naturales, distintos entre sí, sin que el proceso de operación de la máquina sufra alteración alguna.

En el programa vamos a considerar sólo máquinas cuyos estados estén numerados por 0,1,2,..., etc. Siendo *CERO* el estado inicial, lo cual (como veremos) no constituye una limitación.

Si una línea de la tabla que define a M empieza con ja_iP, entonces j es llamado *un estado terminal*.

Ejemplos:

1.- El ejemplo que mencionamos, construir una Máquina de Turing cuyo trabajo al ponerla en operación sobre cualquier cuadro de cualquier expresión de cinta sea moverse un cuadro a la derecha y detenerse, queda descrito por la siguiente tabla:

0	a_0	D	1
0	a_1	D	1
⋮	⋮	⋮	⋮
0	a_n	D	1
1	a_0	P	1
1	a_1	P	1
⋮	⋮	⋮	⋮
1	a_n	P	1

a esta máquina la llamaremos D.

2.- De manera análoga podemos definir la máquina Izquierda (I) usando la siguiente tabla

0	a_0	I	1
0	a_1	I	1
⋮	⋮	⋮	⋮
0	a_n	I	1
1	a_0	P	1
1	a_1	P	1
⋮	⋮	⋮	⋮
1	a_n	P	1

La máquina izquierda colocada sobre cualquier cuadro de cualquier expresión de cinta retrocede un cuadro hacia la izquierda y se detiene. La expresión de cinta no es alterada al aplicar las máquinas (I) o (D).

Comentario: Nótese que una vez que aparece la instrucción de parada P (en algún renglón de la matríz) es irrelevante el estado que se pone a la derecha de P en ese renglón. Sin embargo, debemos rellenar la tabla de alguna manera para satisfacer la definición.

3.- La máquina a_j ($j=0,\ldots,n$) está descrita por la siguiente tabla

0	a_0	a_j	0
0	a_1	a_j	0
⋮	⋮	⋮	⋮
0	a_j	P	0
⋮	⋮	⋮	⋮
0	a_n	a_j	0

esta máquina colocada sobre cualquier cuadro de cualquier expresión de cinta imprime el símbolo a_j y se detiene.

De esta forma hemos definido las máquinas elementales D, I, a_j (con $j=0,1,\dots,n$), a partir de las cuales construiremos máquinas más complicadas.

4.- Las máquinas de los ejemplos que a continuación se consideran son máquinas sobre el alfabeto unitario { / }. Se conviene en representar al símbolo vacío mediante el símbolo ✶ . Se trata, así, de máquinas sobre Números Naturales. Nótese la correspondencia: 0 ⟶ / , 1 ⟶ // , 2 ⟶ ///... etc. Examinaremos cómo trabajan estas máquinas en casos especiales.

(i)

0	✶	/	0
0	/	D	1
1	✶	P	1
1	/	P	1

Si la máquina M , definida por esta tabla, es aplicada en el primer cuadro en blanco a la derecha del último trazo de una secuencia de trazos que han sido escritos sobre la cinta (previamente vacía), entonces M adhiere un trazo a la secuencia y se detiene a la derecha de este último trazo. De este modo, M calcula a la función sucesor.

Nota: Es importante observar (en este ejemplo) que los primeros tres renglones de la tabla son esenciales; no siendo así el cuarto renglón, el cual se escribe sólo para satisfacer la definición de la máquina M.

(ii)

0	*	P	0
0	/	I	1
1	*	P	0
1	/	I	2
2	*	/	3
2	/	I	1
3	*	P	0
3	/	P	0

Si la máquina M, definida por esta tabla, es aplicada sobre el último trazo de una palabra ρ (que ha sido escrita sobre la cinta previamente vacía) entonces M deja de operar después de un número finito de pasos sobre el símbolo (*) o sobre (/) según sea que ρ represente a un número natural par o a un número natural impar respectivamente.

Nota: Como el lector habrá observado, la tabla describe un algoritmo que un calculista humano puede ejecutar. Basta para ello seguir los procedimientos que en el renglón se indican, de acuerdo al estado y el símbolo que haya en la ventana. Esto significa, entre otras cosas, que un calculista humano es una Máquina de Turing.

Como se puede apreciar, toda máquina de Turing opera en forma discreta. Para cambiar de un momento activo al que le sigue la máquina efectúa un ACTO consistente en tres operaciones:

PRIMERA OPERACION

(i) Condición de ventana : Blanco. Imprime alguno de los símbolos $a_1, \ldots, a_n$, o deja el cuadro en la condición en que se encuentra.

(ii) Condición de ventana: Hay un símbolo impreso. Borra el símbolo impreso e imprime alguno de los símbolos $a_1, \ldots, a_n$, o borra el símbolo impreso y deja el cuadro en blanco, o deja el cuadro en la condición en que se encuentra.

SEGUNDA OPERACION

(i) Mueve la ventana de modo que el cuadro en inspección en el siguiente momento sea el que se encuentra inmediatamente a la derecha (izquierda) del actual cuadro en inspección.

(ii) Deja la ventana donde está, es decir permanece en el mismo cuadro en inspección.

TERCERA OPERACION

(i) Cambia a otro estado.

(ii) Permanece en el mismo estado.

Comentario: En las siguientes secciones vamos a considerar funciones B, definidas para todo número entero x, tales que $B(x) \in \{a_0, a_1, \ldots, a_n\}$. En ellas los argumentos x representan los números de los cuadros de la cinta (sobre la cual se reralizarán los cálculos). La cinta esta acomodada en un sentido tal que el cuadro con el número n (también llamado cuadro n) esta inmediatamente a la izquierda del cuadro n+1. Aceptamos que el cuadro n tiene al símbolo $B(n)$ escrito en él. Un cuadro que tenga al símbolo a_0 o $*$ es llamado vacío. Así, podemos considerar la función B como *una expresión de cinta*. Ahora bien, vamos a considerar sólo funciones tales que $B(x) = a_0$ *para casi todas las* x. Es decir, asumimos que sólo un número finito de cuadros tienen algún símbolo (distinto de a_0) escrito en ellos. Por ejemplo:

Si $A = \{a_1, a_2, a_3, a_4\}$; sea $B(-3) = a_2$, $B(-1) = a_4$,

$B(0) = a_3$, $B(2) = a_3$, $B(3) = a_1$ y

$B(x) = *$ para toda $x \in \mathbb{Z}$ tal que $x \neq -3,-1,0,2,3$.

De modo que el aspecto de la cinta es

. . .	$*$	a_2	$*$	a_4	a_3	$*$	a_3	a_1	$*$	. . .
	-4	-3	-2	-1	0	1	2	3	4	

IV.2 Configuraciones.

Por una configuración K de una máquina de Turing M entenderemos cualquier tercia ordenada (A,B,C) donde A es un cuadro (dado por su número), B es una expresión de cinta (es decir, una función) y C es un estado de M.

Una configuración $K = (A,B,C)$ es llamada *inicial* si $C = C_m$. Cualquier configuración $K = (A,B,C)$ corresponde, en forma unívoca, a la línea de la tabla de M que empieza con $CB(A)$. A ésta le llamamos la línea de la configuración K.

Una configuración $K = (A,B,C)$ es llamada *terminal* si la línea de la configuración K empieza con $CB(A)P$.

Sea $K = (A,B,C)$ una configuración no terminal. Sea $CB(A)bc'$ la línea de la configuración K (de modo que $b \neq P$). Podemos asociar (sin ambigüedad) con K *una configuración consecutiva* $F(K) = (A',B',C')$, en la cual tenemos que

$$A' = \begin{cases} A & , \text{ si } \quad b \neq D \quad y \quad b \neq I . \\ A+1 & , \text{ si } \quad b = D . \\ A-1 & , \text{ si } \quad b = I . \end{cases}$$

$$B'(x) = \begin{cases} B(x) & , \text{ si } \quad x \neq A . \\ B(x) & , \text{ si } \quad x = A \quad y \quad (b = D \quad o \quad b = I) . \\ b & , \text{ si } \quad x = A \quad y \quad b \neq D \quad y \quad b \neq I . \end{cases}$$

$$C' = c' .$$

Ahora bien, si tenemos una máquina M, un número A y una función B, entonces una cierta configuración inicial $K_0=(A,B,C_m)$ la caracterizamos diciendo que *aplicamos* M *sobre el cuadro* A *siendo la expresión de cinta* B. Si K_0 no es una configuración terminal, entonces existe para K_0 una *única* configuración consecutiva $K_1 = F(K_0)$. En cuyo caso decimos que M cambia en el primer paso de K_0 a K_1. Si K_1 no es una configuración terminal, entonces existe una *única* $K_2 = F(K_1)$, y decimos que M cambia en el segundo paso de K_1 a K_2, etc. Ahora bien, tenemos dos casos. Ninguna de las configuraciones K_0, K_1, K_2,... es una configuración terminal (y entonces K esta definida para cada n y M nunca deja de operar) o existe una n tal que K_n es una configuración terminal. En este último caso K_{n+1} no esta definida y decimos que M deja de operar después de n pasos (puede ser que n = 0), más aún, si $K_n = (A_n, B_n, C_n)$ decimos que M deja de operar estando sobre el cuadro A_n y siendo B_n la expresión de cinta. La última expresión de cinta B_n y el último cuadro en inspección A_n determinan la letra $B_n(A_n)$ que está escrita en A_n al final del proceso.

IV.3 Equivalencia de Máquinas de Turing.

Decimos que una Máquina de Turing M_1 es *EQUIVALENTE* a una máquina de Turing M_2 si existe una función φ (uno a uno) de los estados de M_1 a los estados de M_2 tal que

1) Cada línea $cabc'$ de M_1 se transfiere en la línea $\varphi(c)ab\varphi(c')$ de M_2; y

2) $\varphi(c_{m1}) = c_{m2}$. { donde c_{m1} es el estado inicial de M_1 y c_{m2} es el estado inicial de M_2 }.

Dada cualquier máquina M_1 podemos siempre hallar una máquina equivalente M_2 en la cual los estados estén numerados por 0, 1, 2,..., etc. y en la cual *CERO* sea el estado inicial. Y viceversa, dada una máquina M_2 cuyos estados estén numerados por 0, 1, 2,..., etc. podemos siempre hallar un máquina de Turing equivalente a M_2 cuyos estados estén numerados con números naturales cualesquiera distintos entre sí. Esto es muy conveniente en relación a la combinación de máquinas que veremos más adelante.

Ahora bien, supongamos que la máquina de Turing M_1 (aplicada sobre el cuadro A siendo la expresión de cinta B) producirá la sucesión de configuraciones $(A_n, B_n(x), C_n)$. Si además suponemos que M_2 es equivalente a M_1, en virtud del mapeo φ de los estados de M_1 a los estados de M_2, entonces podemos observar que la sucesión de configuraciones de M_2 (al ser aplicada sobre el mismo cuadro A y la misma expresión de cinta B) es $(A_n, B_n(x), \varphi(C_n))$. De modo que *máquinas de Turing equivalentes producen la misma sucesión* $(A_n, B_n(x))$ al ser aplicadas sobre el mismo cuadro A y la misma expresión de cinta B.

IV.4 Definición precisa de los conceptos constructivos por medio de Máquinas de Turing.

En esta parte damos las definiciones exactas de los conceptos de Calculabilidad y Decidibilidad. Llamando a estos conceptos, de manera más precisa, Turing-Calculabilidad y Turing-Decidibilidad.

Además nos convenceremos de que estos conceptos exactos son reemplazos naturales de los conceptos intuitivos descritos en la sección II.1 (asumiendo la correspondencia entre Máquinas de Turing y Algoritmos).

En las consideraciones siguientes tomamos como fijo el Alfabeto $A = \{a_1, \ldots, a_n\}$. Sólo consideramos *palabras novacías* sobre este alfabeto, lo cual no representa una limitación (según se estableció en la sección I.3).

Turing-Calculabilidad.

Primero se establece la definición en el caso especial de funciones singulares (es decir, funciones de un argumento).

Sea f una función singular que esta definida para todas las palabras de A y cuyos valores son palabras de A. Decimos que f *es Turing-Calculable* si existe una máquina de Turing M sobre A tal que si escribimos una palabra arbitraria p (sobre A) en la cinta vacía y si aplicamos M sobre un cuadro arbitrario de la cinta, entonces M deja de operar (después de un número finito de pasos) tras haber calculado el valor $f(p)$ de la función. Es decir, sobre el primer cuadro en blanco a la derecha de $f(p)$.

Ahora bien, si $f(p_1,\ldots,p_k)$ es una función de k argumentos, los cuales al igual que los valores de la función son palabras sobre A, entonces decimos que f es *Turing-Calculable* si existe una máquina de Turing M sobre A tal que si escribimos (sobre la cinta vacía) k argumentos arbitrarios $p_1,\ldots,p_k$, en este orden y dejando un cuadro vacío entre las palabras (es decir, $p_1 * p_2 \cdots * p_k$), y si aplicamos M sobre un cuadro arbitrario de la cinta, entonces M deja de operar (después de un número finito de pasos) tras haber calculado el valor $f(p_1,\ldots p_k)$ de la función. Es decir, sobre el primer cuadro en blanco a la derecha de $f(p_1,\ldots,p_k)$.

Comentario: Se puede pensar que la clase de las Funciones Turing-Calculables definidas de este modo es más pequeña que aquella en la cual la máquina (requerida para llevar a cabo el cálculo) se aplica sobre un cuadro particular (para nuestros fines, sobre el primer cuadro vacío a la derecha de los argumentos). Este no es el caso según se demuestra en el Teorema 1 del Apéndice B.

Es admisible, por razones sistemáticas, hablar también de funciones de *cero argumentos*. El valor de una función de *dos argumentos* se puede determinar si damos ambos argumentos. El valor de una función de *un argumento* se puede determinar si damos dicho argumento. De modo similar, el valor de una función de *cero argumentos* se puede determinar si no tenemos argumento alguno. Por lo que, *una función de cero argumentos tiene sólo un valor*. Este puede ser una palabra arbitraria p. Vamos a denotar a la función de cero argumentos cuyo valor es p por C_0^p.

Ahora bien, una función de cero argumentos es Turing-Calculable si existe una máquina de Turing que al aplicarla sobre la cinta vacía deja de operar, después de un número finito de pasos, tras haber calculado el valor de la función. Es decir, sobre el primer cuadro en blanco a la derecha de dicho valor. Es claro que cualquier función de cero argumentos es Turing-Calculable. Puesto que para calcular C_0^p sólo necesitamos tomar una máquina de Turing que escriba la palabra ρ sobre la cinta e inmediatamente deje de operar.

Turing-Decidibilidad.

Sea P_0 un conjunto de palabras sobre A. Decimos que P_0 es Turing-Decidible si existe una máquina de Turing M sobre A y dos símbolos distintos a_i, $a_j \in A \cup \{a_0\}$ tales que si escribimos una palabra arbitraria ρ ($\rho \in P_A$) sobre la cinta vacía y aplicamos M sobre un cuadro arbitrario de la cinta, entonces M se detiene después de un número finito de pasos sobre a_i o a_j según sea que $\rho \in P_0$ o $\rho \notin P_0$ respectivamente. En tal caso, se dice que M decide a P_0 con la ayuda de a_i y a_j.

Sean P_1 y P_2 conjuntos de palabras sobre A, tales que $P_2 \subset P_1$. Decimos que P_2 es Turing-Decidible con respecto a P_1 si existe una máquina de Turing M sobre A y dos símbolos distintos a_i, $a_j \in A \cup \{a_0\}$ tales que si escribimos una palabra arbitraria ρ ($\rho \in P_1$) sobre la cinta vacía y aplicamos M sobre un cuadro arbitrario de la cinta, entonces M se detiene después de un número finito de pasos sobre a_i o a_j según sea que $\rho \in P_2$ o $\rho \notin P_2$ respectivamente. En tal caso, se dice que M decide a M_2 con respecto a M_1 con la ayuda de a_i y a_j.

Tanto en el caso de Decidibilidad "Absoluta" como en el de Decidibilidad "con respecto a", los símbolos particulares a_i y a_j son irrelevantes. A continuación probamos este hecho para el caso de Decidibilidad Absoluta.

Demostraremos que: si M decide a P_0 con la ayuda de a_i y a_j, entonces si a_k y $a_l \in A \cup \{a_0\}$ existe una máquina M' sobre A que decide a P_0 con la ayuda de a_k y a_l.

Tómese la tabla de M y añádase un nuevo estado R. Ahora bien, cada línea de M con la forma ea_iPe' se cambia por ea_ia_kR, y cada línea de M con la forma ea_jPe' se cambia por ea_ja_lR. Finalmente añádanse n+1 líneas de la forma Ra_hPR (con $0 \leq h \leq n$). Así, cada vez que M se detiene sobre a_i o a_j nuestra nueva máquina M' se detiene sobre a_k o a_l respectivamente.

Definición. Se dice que una Propiedad (o Predicado) de palabras es Turing-Decidible si el conjunto de palabras con esta propiedad es Turing-Decidible.

Definimos la Turing-decidibilidad de Relaciones n-arias ($n \geq 2$) de un modo similar. Es decir, una relación n-aria ($n \geq 2$) es Turing-decidible si el conjunto de n-adas que están en la relación es Turing-decidible.

IV.5 Combinación de Máquinas de Turing.

Debido a que en ocasiones se dificulta interpretar el funcionamiento de una máquina representada por una tabla larga, es recomendable la introducción de operaciones para ayudarnos, de manera que podamos combinar tablas simples en otras más complicadas. El objeto de esta sección es establecer que cualquier máquina de Turing sobre un alfabeto $\{a_1, \ldots, a_n\}$ se puede construir a partir de las máquinas elementales D, I, a_j (con $0 \leq j \leq n$) introducidas en la sección IV.1 . El modo que se empleará para combinarlas es análogo al que se utiliza para construir los «diagramas de flujo» utilizados en la programación de computadoras electrónicas.

Diagramas.

Considérense r máquinas que numeramos $M_1,\dots,M_r$. Todas ellas sobre un alfabeto fijo $\{a_1,\dots,a_n\}$. Un *Diagrama* es una *Digráfica* finita cuyos vértices se han numerado con los r números (pudiendo haber repeticiones, es decir más de r vértices) de modo que:

1) Al menos un vértice corresponde a M_i $(1 \le i \le r)$.

2) Uno de los vértices es marcado como *vértice inicial* (encerrándolo en un círculo).

3) Cada arista de la Digráfica esta marcada con un número j, con $0 \le j \le n$.

4) De cada vértice sale a lo más una arista marcada con el símbolo j, con $0 \le j \le n$.

Harémos uso de las siguientes abreviaturas:

$$M \longrightarrow T \quad \text{en vez de} \quad M \begin{array}{c} \xrightarrow{0} \\ \vdots \\ \xrightarrow{n} \end{array} T .$$

$$M \xrightarrow{\neq j} T \quad \text{en vez de} \quad M \begin{array}{c} \xrightarrow{0} \\ \vdots \\ \xrightarrow{n} \end{array} T \quad \text{salvo la arista } j .$$

$$M^2 \quad \text{en vez de} \quad M \longrightarrow M .$$

$$M^n \quad \text{en vez de} \quad M^{n-1} \longrightarrow M .$$

$$MT \quad \text{en vez de} \quad M \longrightarrow T .$$

Por M^O o M_O entenderemos la máquina que está dada por la siguiente tabla:

O	a_o	P	O
O	a_1	P	O
⋮	⋮	⋮	⋮
O	a_n	P	O

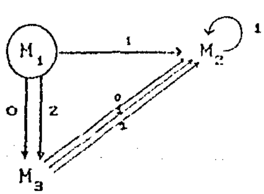
Es decir, el único acto de operación de esta máquina es detenerse[1].

Comentario: Si a un vértice no llegan aristas entonces ese vértice es el inicial. En tal caso se omite el círculo. Si el vértice inicial no se marca, se considera tál el que se encuentra en el extremo superior izquierdo del diagrama.

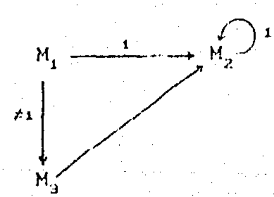
Ejemplo:

Si $n = 2$, entonces

la digráfica



se abrevia



(1) En el PROGRAMA a esta Máquina la llamamos M0.

Definición de la máquina de Turing M representada por una digráfica D.

Obtenemos una tabla para M de la siguiente manera:

1.- Se toma una tabla para cada máquina representada en el diagrama. Cuando una máquina aparece más de una vez en el diagrama se toma una tabla *diferente* por cada ocurrencia. Para ello se conviene en lo siguiente : *no hay dos tablas separadas que contengan estados iguales* . Es importante mencionar que según la definición de Máquina de Turing los estados han de numerarse desde 0 hasta M-1 . Sin embargo, si aparece una máquina varias veces, por cada ocurrencia de ésta se escribirá la tabla de una máquina equivalente a ella de modo que no existan dos tablas separadas que contengan un mismo estado.

2.- Con las tablas singulares se forma una tabla $\overline{M}$ colocándolas una tras otra en cualquier orden. La única excepción es que la tabla de la máquina que esta asociada con el vértice inicial se coloca en primer término

3.- Si en el diagrama aparece la combinación de símbolos $M_i \xrightarrow{k} M_j$, entonces cada línea de la forma ea_kPe' de la tabla de M_i se transforma en la línea $ea_ka_ke_{mj}$, donde e_{mj} denota al estado inicial de la máquina M_j.

Llevando a cabo el paso (3) la tabla $\overline{M}$ se transforma en una tabla M para la máquina representada por D. La idea es la siguiente: si echamos a andar M, siendo la expresión de cinta B(X) y el cuadro en inspeccion A, entonces M ejecuta primero los mismos pasos de la máquina M' (la cual denota a la máquina inicial) hasta que M' deja de operar sobre una cierta configuración (A_n, B_n, C_n). En este caso la línea de la forma

$$ea_k Pe' \qquad (i)$$

es decisiva para M'. Es posible que M' no esté conectada con alguna otra máquina, entonces la línea (i) no es alterada y M también deja de operar. Sin embargo, si aparece la conexión $M' \xrightarrow{k} M''$ entonces M ejecuta la línea $ea_k a_k e_{m''}$ en vez de la línea (i). Es decir, cada vez que M' se va a detener (estando en la ventana el símbolo a_k) en la máquina correspondiente al diagrama se pasa a operar la tabla correspondiente a M'', teniendo en la ventana el símbolo a_k. En este caso, la expresión de cinta y el cuadro inspeccionado no se alteran, sólo el estado inicial de M'' es puesto en acción. En otras palabras, $A_{n+1}=A_n$, $B_{n+1}=B_n$ y $C_{n+1}=e_{m''}$.

Cabe decir que la máquina M efectúa, en orden sucesivo, el trabajo de las máquinas M', M'', ..., en una secuencia determinada por el Diagrama junto con la expresión de cinta original y el cuadro inspeccionado al accionar la máquina.

Ejemplo:

En la sección anterior introducimos las funciones C_0^k de cero argumentos, las cuales tienen el valor constante K. Podemos, en general, considerar para cualquier $n \in \mathbb{N}$ una función de n argumentos que tiene el valor constante K (K $\in \mathbb{N}$). Todas estas funciones C_n^k son Turing-Calculables. El cálculo puede ser llevado a cabo por la máquina

$$* \, (D/)^{k+1} \, D \, * \, .$$

Usando $*$ producimos un cuadro vacío, el cual marca el inicio del valor de la función por calcular. Ahora bien, usando $(D/)^{k+1}$ escribimos el valor K de la función C_n^k. Finalmente al usar $D \, *$ determinamos la existencia de un cuadro vacío a la derecha del valor de las función.

Definición.

Dos máquinas M y M' son Intercambiables si para cualquier expresión de cinta B y cualquier cuadro inicial A sucede que: Si M (al ser aplicada sobre el cuadro A siendo la expresión de cinta B) alcanza una configuración terminal (A_n, B_n, C_n), entonces M' (al ser aplicada sobre el mismo cuadro A siendo la expresión de cinta B) alcanza una configuración terminal (A_r, B_r, C_r) con $A_n = A_r$ y $B_n = B_r$, y viceversa.

PROPOSICION FUNDAMENTAL :

Para toda máquina de Turing M sobre el alfabeto $\{a_1, \ldots, a_n\}$ es posible dar efectivamente una combinación de las máquinas elementales $D, I, a_0, a_1, \ldots, a_n$ la cual es intercambiable con M.

La demostración es por diagramas. Cada línea de la tabla de M da lugar a un fragmento del diagrama, interviniendo en él sólo máquinas elementales, del modo siguiente:

Si la línea de la tabla de M es		el fragmento del diagrama correspondiente deberá ser:	
$ea_i a_j e'$	:	(vértice) $\xrightarrow{i}$ a_j $\longrightarrow$	sale sólo una arista, correspondiente al renglón de la tabla de M que empieza con $e'a_j$.
$ea_i De'$	:	(vértice) $\xrightarrow{i}$ D $\longrightarrow$ $\vdots$ $\longrightarrow$	salen (n+1) aristas correspondientes a los (n+1) renglones del estado e'.
$ea_i Ie'$	:	(vértice) $\xrightarrow{i}$ I $\longrightarrow$ $\vdots$ $\longrightarrow$	salen (n+1) aristas correspondientes a los (n+1) renglones del estado e'.
$ea_i Pe'$	:	(vértice) $\xrightarrow{i}$ M^0.	

Siendo el vértice inicial el que representa a M^0.

Ahora bien, debido a que el número de estados de la máquina M es finito y el número de letras del alfabeto también es finito, tenemos que el diagrama es finito. Además, si la línea correspondiente a $e'a_j$ ya se consideró entonces la arista termina en un vértice ya trazado.

Observación: *La máquina así definida no es la más reducida que es Intercambiable* con M.

Ejemplo:

Escogemos la tabla de la máquina de Turing que representa a la función sucesor (la cual mencionamos en la sección IV.1). Dicha tabla, a la cual llamaremos SUC , esta definida de la siguiente manera:

0	*	/	0
0	/	D	1
1	*	P	1
1	/	P	1

Ahora bien, usando *La Proposición Fundamental*, es posible dar una combinación de las máquinas elementales que sea *intercambiable* con la máquina que define la tabla anterior. El diagrama correspondiente a dicha combinación es

$$\begin{tikzcd}
M_0 \arrow[r, "0"] \arrow[dr, "1"'] & / \arrow[d, "1"] & \\
& D \arrow[r, shift left, "0"] \arrow[r, shift right, "1"'] & M_0
\end{tikzcd}$$

Al describir las tablas de las máquinas usadas en el diagrama, se considera que las tablas de las máquinas /, D y M_0 (segunda aparición) han sido reemplazadas por tablas de máquinas *equivalentes* de modo que no existan dos tablas con un mismo estado.

De esta manera, las tablas de las máquinas que intervienen en el Diagrama son:

Máquina M_0

0	*	P	0
0	/	P	0

Máquina /

1	*	/	1
1	/	P	1

Máquina D

2	*	D	3
2	/	D	3
3	*	P	3
3	/	P	3

Máquina M_0 (segunda aparición)

4	*	P	4
4	/	P	4

Ahora construimos la tabla $\overline{M}$, colocando una tras otra las tablas anteriores(1). Y para terminar, construimos la tabla M haciendo los cambios necesarios a $\overline{M}$.

$\overline{M}$

0	*	P	0
0	/	P	0
1	*	/	1
1	/	P	1
2	*	D	3
2	/	D	3
3	*	P	3
3	/	P	3
4	*	P	4
4	/	P	4

M

0	*	*	1
0	/	/	2
1	*	/	1
1	/	/	2
2	*	D	3
2	/	D	3
3	*	*	4
3	/	/	4
4	*	P	4
4	/	P	4

es claro que M es *intercambiable* con SUC.

(1) recordemos que la única restricción al construir M, basándonos en las tablas de las máquinas que intervienen en el diagrama, es que la tabla asociada al vértice inicial sea colocada en primer término.

IV.6 Máquinas de Turing Especiales.

Ahora construiremos, usando los procesos discutidos en la sección anterior, algunas Máquinas de Turing (las cuales necesitaremos más tarde) haciendo uso de las Máquinas elementales D, I, $a_0, \ldots, a_n$ introducidas en la sección IV.1 . Recurrimos para ello a la siguiente notación:

m	Un cuadro marcado.
$\sim$	Un cuadro con o sin marca.
$*$	Un cuadro vacío.
$* \cdots *$	Secuencia finita de cuadros vacíos.
$* \cdots$	Secuencia infinita hacia la derecha de cuadros en blanco.
ρ	Segmento de la cinta sobre el cual se encuentra impresa la palabra ρ. (ρ es una palabra no vacía).
X	Sucesión de palabras $\rho_1 * \rho_2 \cdots * \rho_n$ (no vacías) llamada *Oración*.

Al ilustrar el método de operación de las Máquinas que estudiaremos,

(i) Caracterizaremos el cuadro inspeccionado *subrayándolo*, y

(ii) La expresión $I \Rightarrow F$ significa: la máquina activada en la situación inicial I se detiene en la situación final F.

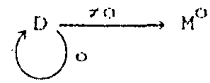
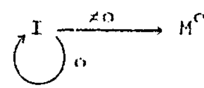
Todas las máquinas descritas a continuación (con excepción posiblemente de S_d, S_i e S) dejan de operar después de un número finito de pasos.

Al ilustrar la estructura de las siguientes máquinas recordemos que el único acto de operación de la máquina M^o es detenerse. Todas las máquinas se consideran sobre el Alfabeto $A = \{a_1, \ldots, a_n\}$.

1.- La máquina Blanco a la derecha B_d (La máquina Blanco a la izquierda B_i) se mueve, desde el cuadro sobre el cual es activada, un cuadro hacia la derecha (izquierda). Si este cuadro esta vacío, entonces la máquina se detiene. Si, por el contrario, el cuadro esta marcado entonces B_d (B_i) se mueve sobre todos los cuadros marcados hacia la derecha (izquierda) hasta encontrar un primer cuadro vacío, sobre el cual deja de operar. En cualquier caso, la expresión de cinta no se altera.

	Métodos de Operación	Estructura
B_d	$\underline{\sim}\ \rho\ * \Rightarrow \sim\ \rho\ \underline{*}$ $\underline{\sim}\ * \Rightarrow \sim\ \underline{*}$	$\begin{tikzcd} D \arrow[r, "o"] \arrow[loop below, "\neq o"] & M^o \end{tikzcd}$
B_i	$*\ \rho\ \underline{\sim} \Rightarrow \underline{*}\ \rho\ \sim$ $*\ \underline{\sim} \Rightarrow \underline{*}\ \sim$	$\begin{tikzcd} I \arrow[r, "o"] \arrow[loop below, "\neq o"] & M^o \end{tikzcd}$

2.- La máquina Símbolo a la derecha S_d (La máquina Símbolo a la izquierda S_i) se mueve, desde el cuadro sobre el cual es activada, un cuadro hacia la derecha (izquierda). Si este cuadro esta marcado, entonces la máquina se detiene. Si, por el contrario, el cuadro esta vacío entonces S_d (S_i) se mueve hacia la derecha (izquierda) hasta encontrar un primer cuadro marcado, sobre el cual deja de operar. La expresión de cinta no es alterada.

Métodos de Operación		Estructura
S_d	$\underline{\sim} * \cdots * m \Rightarrow \sim * \cdots * \underline{m}$ $\underline{\sim}\, m \Rightarrow \sim \underline{m}$	
S_i	$m * \cdots * \underline{\sim} \Rightarrow \underline{m} * \cdots * \sim$ $m\, \underline{\sim} \Rightarrow \underline{m} \sim$	

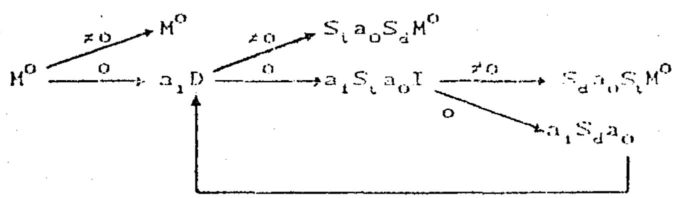
3.- La máquina S realiza la siguiente operación: Si activamos S sobre un cuadro arbitrario de la cinta, sobre la cual al menos un cuadro esta marcado, entonces S dejará de operar (después de un número finito de pasos) sobre un cuadro marcado. La expresión de cinta original coincidirá con la expresión de cinta terminal (durante los cálculos, sin embargo, la expresión de cinta puede ser alterada).

El método de construcción de S está basado en las siguientes observaciones: Si pudieramos asegurar que existe un cuadro marcado a la derecha (izquierda) del cuadro inspeccionado original, entonces alcanzaríamos nuestra meta simplemente usando S_d (S_i). Sin embargo, si existen cuadros marcados (al menos uno) y no sabemos de qué lado del cuadro inspeccionado original está, entonces S busca alternativamente (empezando por la derecha) un primer símbolo sobre el cual dejará de operar. Si el cuadro inspeccionado original está marcado, entonces S ahí mismo deja de operar.

Método de Operación

{ Véase el párrafo anterior }

Estructura



4.- La máquina Límite Derecho L_d (Límite Izquierdo L_i) de una oración realiza la siguiente operación: se mueve desde el cuadro sobre el cual es activada hacia la derecha hasta hallar dos cuadros vacíos consecutivos; dejando de operar sobre el primero de éstos. La expresión de cinta no es alterada.

Métodos de Operación de L_d

$\underline{\sim}\ X * *\quad\Rightarrow\quad \sim X\ \underline{*}\ *$

$\underline{\sim} * X * *\quad\Rightarrow\quad \sim * X\ \underline{*}\ *$

$\underline{\sim} * *\quad\Rightarrow\quad \sim \underline{*}\ *$

Estructura de L_d

$$\begin{tikzcd} B_dD \arrow[loop left, "\neq 0"] \arrow[r, "0"] & I \arrow[r] & M^0 \end{tikzcd}$$

Métodos de Operación de L_ι

$* * X\ \underline{\sim}\quad\Rightarrow\quad * \underline{*}\ X \sim$

$* * X * \underline{\sim}\quad\Rightarrow\quad * \underline{*}\ X * \sim$

$* * \underline{\sim}\quad\Rightarrow\quad * \underline{*} \sim$

Estructura de L_ι

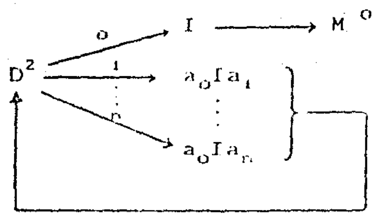
$$\begin{tikzcd} B_\iota I \arrow[loop left, "\neq 0"] \arrow[r, "0"] & D \arrow[r] & M^0 \end{tikzcd}$$

5.- La máquina 1_ι recorre una palabra ρ (letra por letra) un cuadro a la izquierda cuando la ventana se ubica [illegible] a la izquierda de dicha palabra ρ.

Método de Operación

[illegible]

Estructura de 1_i

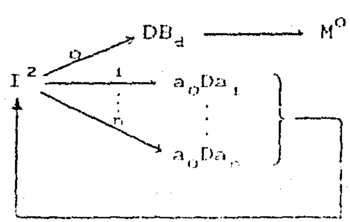


6.- La máquina 1_d recorre una palabra ρ (letra por letra) un cuadro a la derecha cuando la ventana se ubica dos cuadros a la derecha de dicha palabra ρ.

Método de Operación

$$* \rho * \underline{*} \quad \Rightarrow \quad * * \rho \underline{*}$$

Estructura



7.- La máquina Borra y Desplaza (BODE) realiza la siguiente operación: Dadas dos palabras seguidas ($* \rho * q *$) y el cuadro en inspección el que está a la derecha de q, al aplicar BODE ρ es borrada y en su lugar se coloca q. Después de esto BODE deja de operar quedando sobre el cuadro que está a la derecha de la palabra q.

Método de Operación

$$* \rho * q \underline{*} \quad \Rightarrow \quad * q \underline{*} \cdots *$$

Estructura

$$\begin{tikzcd}
B_\iota I \arrow[r, "\neq 0"] \arrow[dr, "0"'] & a_0 1_\iota \arrow[l, bend right, to path={-- ([yshift=3ex]\tikztostart.north) -| (\tikztotarget)}] & \\
& 1_\iota \arrow[r] & M^0
\end{tikzcd}$$

8.- La máquina BORRAR (Br) realiza la siguiente operación: Elimina de la cinta los cálculos secundarios, los cuales suponemos que han sido escritos sobre la cinta en forma de una oración X. Es fundamental la existencia de al menos dos cuadros vacíos antes de X.

Método de Operación

$$* * * X * \rho \underline{*} \quad \Rightarrow \quad * \rho \underline{*} \cdots *$$

Estructura

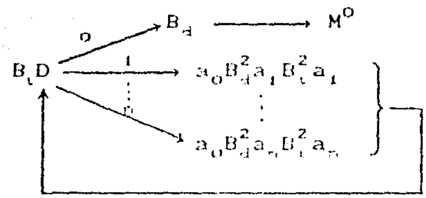
$$\begin{tikzcd}
B_\iota I \arrow[r, "\neq 0"] \arrow[dr, "0"'] & B\rho_\iota (\text{BODE}) \arrow[l, bend right] \\
& 1_\iota B_\iota I_\iota M^0
\end{tikzcd}$$

9.- La máquina C (Copia) realiza la siguiente operación: Vuelve a escribir la última palabra escrita en la cinta.

Método de Operación

$$* \rho \underline{*} \cdots \quad \Rightarrow \quad * \rho * \rho \underline{*} \cdots$$

Estructura

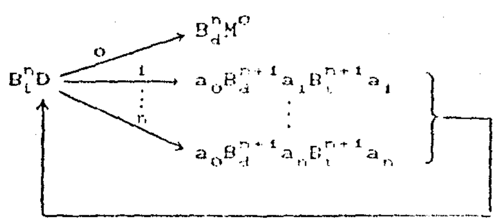


10.- La máquina que denotaremos por C_n realiza la siguiente operación: Copia la n-ésima palabra de derecha a izquierda.

Método de Operación

$$* \rho_n * \rho_{n-1} \cdots * \rho_1 \underline{*} \cdots \quad \Rightarrow \quad * \rho_n * \rho_{n-1} \cdots * \rho_1 * \rho_n \underline{*} \cdots$$

Estructura



Nota: En esta sección hemos utilizado la máquina M^O para indicar el momento en que cada máquina deja de operar. Esto se aplica al construir digráficas en el programa. consultar el apéndice C.

CAPITULO V

FUNCIONES RECURSIVAS Y PREDICADOS RECURSIVOS

La clase de la Funciones Recursivas surge al hacer del concepto de Función Calculable algo más preciso. Ciertas funciones Iniciales (que pueden considerarse como calculables de inmediato) son llamadas *RECURSIVAS*. Para generar nuevas funciones recursivas, a partir de aquellas previamente alcanzadas, se dispone de tres reglas. Cada una de estas reglas indica un Algoritmo para calcular los valores de la nueva función una vez que se han calculado los valores de las funciones que la definen. Consideramos sólo funciones cuyos argumentos y valores son *Números Naturales*. Es decir, *FUNCIONES ARITMETICAS*.

V.1 Funciones Iniciales.

a) La Función Sucesor $S : \mathbb{N} \longrightarrow \mathbb{N}$ de grado 1. Cuyo valor, para cada $n \in \mathbb{N}$, es $S(n) = n+1$.

b) La Función Idéntica $I_{n,k}: \mathbb{N}^n \longrightarrow \mathbb{N}$ de grado n. Donde $1 \leq k \leq n$. La cual esta definida por la ecuacion

$$I_{n,k}(x_1,\ldots,x_n) = x_k \; .$$

c) La Función Constante 0 de grado cero . $0 : \{ 0 \} \longrightarrow \mathbb{N}$. Esta función la denotaremos $I_{0,0}$.

Comentario: Una función aritmética de grado n es una correspondencia $f : \mathbb{N}^n \longrightarrow \mathbb{N}$. Cuando $n = 0$ el dominio es $\mathbb{N}^0 = \{ 0 \}$. En tal caso, la función no es otra cosa que la elección de un elemento de $\mathbb{N}$ con el cual se le identifica. En el caso que aquí se trata, la función es $O(0) = 0$.

V.2 Reglas para generar nuevas funciones.

a) **Composición** .- Si g es una función de grado m ($m \geq 1$) y además $h_1, \ldots, h_m$ son funciones de grado n ($n \geq 0$), entonces la ecuación

$$f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n))$$

define una función de grado n.

b) **Recursión** .- Si g es una función de grado n ($n \geq 0$) y h es una función de grado n+2, entonces el sistema de ecuaciones

$$f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n)$$

$$f(x_1, \ldots, x_n, S(y)) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n))$$

define una función de grado n+1.

c) **Operador** μ .- Si g es una función de grado n+1 con la propiedad de que para cada n-ada de números naturales $(x_1, \ldots, x_n)$ hay un número natural z para el cual $g(x_1, \ldots, x_n, z) = 0$, entonces la ecuación $f(x_1, \ldots, x_n) = \mu z(\ g(x_1, \ldots, x_n, z) = 0\ )$ define una función de grado n . En la ecuación anterior la expresión $\mu z(g(x_1, \ldots, x_n, z) = 0)$ denota al menor número natural z tal que $g(x_1, \ldots, x_n, z) = 0$. A μ se le llama Operador Minimal.

V.3 Definición de Funcion Recursiva.

Una función aritmética es llamada RECURSIVA si es inicial o si se puede generar a partir de las funciones iniciales aplicando un número finito de veces las reglas de Composición, Recursion y/o el Operador μ. Si en la construcción de una función no se hace uso del operador μ , entonces se dice que la función es RECURSIVA PRIMITIVA.

Comentarios.

1.- Una definición alternativa es la siguiente: La clase de las funciones recursivas es la mínima clase que contiene a las funciones iniciales y es cerrada bajo las operaciones de Composición, Recursión y aplicación del operador μ . Si se omite el que la clase sea cerrada bajo el operador μ, entonces la clase es la de las funciones recursivas primitivas.

2.- Es claro que *Toda Función Recursiva es Calculable*, y que *Toda Función Recursiva Primitiva es Recursiva*. Aunque aquí no lo haremos, se puede demostrar que el recíproco de esta última proposición es falso.

Ejemplos.

A continuación se escribe una lista de funciones recursivas junto con su definición.

1.- $S(x)$ (función inicial).

2.- $I_{n,k}(x_1,\ldots,x_n)$ (funciones iniciales).

3.- $I_{0,0}$ (función inicial).

4.- La función suma

$$+(x,0) = I_{1,1}(x)$$

$$+(x,S(y)) = S(I_{3,3}(x,y,+(x,y))).$$

De aquí en adelante, escribiremos las ecuaciones que definen cada función en notación ordinaria y sin incurrir en detalles. En particular, omitiremos la escritura de algunas funciones iniciales, algunas composiciones y algunas recursiones. Por ejemplo, en vez de la escritura utilizada al definir la función suma, simplemente escribiremos

$$x + 0 = x$$

$$x + Sy = S(x + y).$$

El lector notará que esto no representa problema alguno.

5.- La función producto

$$x\cdot 0 = 0$$

$$x\cdot Sy = x\cdot y + x.$$

6.- Las funciones Constantes.

La función $I_{0,0}$ es de grado cero. Al componerla con la función sucesor se genera una función de grado cero que corresponde al número 1 . Aplicando reiteradamente este procedimiento se generan en forma consecutiva los números naturales:

$$\begin{array}{lll} 0 & = & C_0^0 \text{ (función inicial)} \\ 1 & = & S(0) \\ 2 & = & S(S(0)) \\ 3 & = & S(S(S(0))) \\ \vdots & \vdots & \vdots \end{array}$$

7.- La función elevar a una potencia

$$x^0 = 1$$

$$x^{Sy} = x^y \cdot x \quad .$$

8.- La función factorial

$$0! = 1$$

$$(Sx)! = x! \cdot Sx \quad .$$

9.- La función predecesor ($pd(x)$)

$$pd(x) = \begin{cases} 0 & \text{si} \quad x=0 \\ x-1 & \text{si} \quad x\neq 0 \end{cases}$$

esta definida en forma recursiva por

$$pd(0) = 0$$

$$pd(Sx) = x.$$

10.- La diferencia positiva de x con y ($x \dot{-} y$)

$$x \dot{-} y = \begin{cases} 0 & \text{si} \quad x < y \\ x-y & \text{si} \quad y \leq x \end{cases}$$

esta definida en forma recursiva por

$$x \dot{-} 0 = x$$

$$x \dot{-} Sy = pd(x \dot{-} y).$$

11.- La diferencia absoluta ($|x - y|$)

$$|x - y| = \begin{cases} x-y & \text{si} \quad y \leq x \\ y-x & \text{si} \quad x < y \end{cases}$$

esta definida en forma recursiva por

$|x - y| = (x \dot{-} y) + (y \dot{-} x)$.

12.- La función signo (sg(x))

$$sg(x) = \begin{cases} 0 & \text{si} \quad x = 0 \\ 1 & \text{si} \quad x > 0 \end{cases}$$

esta definida en forma recursiva por

$sg(0) = 0$, $sg(S(x)) = 1$.

13.- La función signo contrario ($\overline{sg}(x)$)

$$\overline{sg}(x) = \begin{cases} 1 & \text{si} \quad x = 0 \\ 0 & \text{si} \quad x > 0 \end{cases}$$

esta definida en forma recursiva por

$\overline{sg}(0) = 1$, $\overline{sg}(S(x)) = 0$.

14.- La función

$$E(x,y) = \begin{cases} 0 & \text{si} \quad x = y \\ 1 & \text{si} \quad x \neq y \end{cases}$$

esta definida en forma recursiva por

$E(x,y) = sg(|x - y|)$.

15. - La función

$$F(x,y) = \begin{cases} 1 & \text{si} \quad x = y \\ 0 & \text{si} \quad x \neq y \end{cases}$$

está definida en forma recursiva por

$$F(x,y) = \overline{sg}(|x - y|).$$

Para terminar esta sección demostramos una proposición que usaremos más adelante, en ella X denota a $x_1, \ldots, x_n$.

Proposición. Si $f(X,k)$ es una función recursiva primitiva, entonces las funciones

$$\text{(i)} \quad \varphi_f(X,z) = \sum_{k=0}^{z} f(X,k) \qquad \text{(suma acotada de } f\text{)}$$

$$\text{y} \quad \text{(ii)} \quad \psi_f(X,z) = \prod_{k=0}^{z} f(X,k) \qquad \text{(producto acotado de } f\text{)}$$

son recursivas primitivas [1].

Demostración.

(i) $\varphi_f(X,0) = f(X,0)$

$$\varphi_f(X,S(z)) = \varphi_f(X,z) + f(X,S(z)).$$

(ii) $\psi_f(X,0) = f(X,0)$

$$\psi_f(X,S(z)) = \psi_f(X,z) \cdot f(X,S(z)).$$

(1) Estas funciones son de grado n+1 igual que f. Sin embargo, dependen de X y z, mientras que f depende de X y de k.

V.4 Predicados Recursivos.

En la sección anterior hemos mostrado la recursividad de algunas funciones. En los ejemplos dados al final, hemos podido hacer esto substituyendo las definiciones originales (de las funciones en cuestión) por otras equivalentes de modo que la recursividad de éstas sea evidente. En esta sección introducimos los conceptos de Predicado Recursivo y Predicado Recursivo Primitivo.

Definición (de Predicado).

Un Predicado n-ario ($n \geq 1$) es una relación n-aria, entre números naturales, que es válida para ciertas n-adas(ordenadas) de números. El predicado "ser número primo", por ejemplo, es un predicado singular que es válido para 2,3,5,..., y no es válido para 4,6,8,.... La relación "menor que" es un predicado binario que es válido para el par ordenado (5,9), pero no es válido para el par (7,2) o el par (3,3). Podemos también, por ejemplo, considerar el predicado ternario de "estar entre" el cual es válido para la tercia (4,6,9), puesto que $4 < 6 < 9$.

Sea $\bar{w} = (w_1, \ldots, w_n)$ un elemento de $\mathbb{N}^n$. Si escribimos $P\bar{w}$ o $\bar{w} \in P$ significará que el predicado P es válido para la n-ada $\bar{w}$. Y si escribimos $\bar{w} \notin P$ significará que el predicado P no es válido para la n-ada $\bar{w}$.

Definición (de Predicado Recursivo y de Predicado Recursivo Primitivo).

Un Predicado n-ario P ($n \geq 1$) es llamado Recursivo (Recursivo Primitivo) si existe una función n-aria f Recursiva (Recursiva Primitiva) tal que, para toda n-ada $\bar{w}$ de números,

$$\bar{w} \in P \quad \text{si y sólo si} \quad f(\bar{w}) = 0.$$

Por lo tanto, podemos determinar, calculando el valor de f para el argumento $\bar{w}$, si P es válido o no para $\bar{w}$. Esto demuestra, puesto que cualquier función recursiva (recursiva primitiva) es calculable, que todo predicado recursivo (recursivo primitivo) es absolutamente decidible.

En la sección II.2 introducimos el concepto de función característica de un conjunto. Ahora hablaremos, más generalmente, de función característica de un Predicado.

Definición.

La función n-aria f es llamada Función Característica del Predicado n-ario P si y sólo si para toda $\bar{w}$

$$(\bar{w} \in P \iff f(\bar{w}) = 0) \text{ y } (\bar{w} \notin P \iff f(\bar{w}) = 1).$$

Cualquier Predicado tiene exactamente una función característica. En seguida, tenemos un resultado importante.

Teorema. Un Predicado P es Recursivo (Recursivo Primitivo) si y sólo si la función característica de P es Recursiva (Recursiva Primitiva).

Para probar esto, sólo necesitamos demostrar que la función característica de un Predicado Recursivo (Recursivo Primitivo) P es Recursiva (Recursiva Primitiva).

Por hipótesis, como P es Recursivo (Recursivo Primitivo), existe una función f Recursiva (Recursiva Primitiva), que depende de P, tal que

$$P\bar{w} \iff f(\bar{w}) = 0 \qquad \text{para toda } w$$

Sea $h(\bar{w}) = sg(f(\bar{w}))$. Entonces h es Recursiva (Recursiva Primitiva) y es precisamente la función característica de P.

Veamos ahora algunos ejemplos de Predicados Recursivos:

1) $x \leq y$. En este caso la función $f(x,y) = x \dot{-} y$ es una función recursiva primitiva tal que

$x \leq y$ si y sólo si $f(x,y) = 0$ para todo $(x,y) \in \mathbb{N}^2$.

2) $x < y$. Sea $f(x,y) = Sx \dot{-} y$, por lo tanto,

$x < y$ si y sólo si $f(x,y) = 0$ para todo $(x,y) \in \mathbb{N}^2$.

3) $x = y$. La función característica de este predicado es $f(x,y) = sg(|x-y|)$, la cual, como vimos en la sección anterior, es recursiva primitiva.

4) $x \geq y \iff y \leq x$.

5) $x > y \iff y < x$.

6) $x \,/\, y \iff \overset{y}{\underset{z=0}{\exists}}\, xz = y$ $\quad$ { $\overset{x}{\underset{z=0}{\exists}}$ significa "que existe z entre 0 y x incluyendo 0 y x tal que"}

7) $Mx \iff 2 \,/\, x$. $\quad$ {Mx significa "x es par"}.

8) $Nx \iff \neg Mx$. $\quad$ {Nx significa "x es impar"}

{El predicado $\neg Mx$ es Recursivo, ver la proposición de la página siguiente}.

9) $Pr(x) \iff x \neq 0 \wedge x \neq 1 \wedge \overset{x}{\underset{z=0}{\forall}} (z/x \Rightarrow z = 1 \vee z = x)$.

{$Pr(x)$ significa "x es primo"}

{ $\overset{x}{\underset{z=0}{\forall}}$ significa "que para toda z entre 0 y x incluyendo 0 y x"}.

Definición.

Sean $R\bar{w}$ y $S\bar{w}$ predicados recursivos n-arios. Los predicados $\neg R$, $R \wedge S$, $R \vee S$, son los siguientes:

(i) $\bar{w} \in \neg R$ si y sólo si $\bar{w} \notin R$.

(ii) $\bar{w} \in R \wedge S$ si y sólo si $\bar{w} \in R$ y $\bar{w} \in S$.

(iii) $\bar{w} \in R \vee S$ si y sólo si $\bar{w} \in R$ o $\bar{w} \in S$.

Proposición 1.

Sean $R\bar{w}$ y $S\bar{w}$ predicados recursivos (recursivos primitivos) n-arios. En tal caso:

(i) $\neg R\bar{w}$ es recursivo (recursivo primitivo).

(ii) $R\bar{w} \wedge S\bar{w}$ es recursivo (recursivo primitivo).

(iii) $R\bar{w} \vee S\bar{w}$ es recursivo (recursivo primitivo).

Demostración.

Por hipótesis, las funciones $C_r(\bar{w})$ y $C_s(\bar{w})$ son recursivas (recursivas primitivas).

(i) la función característica de $\neg R$ es $\overline{sg}\,(C_r(\bar{w}))$.

(ii) la función característica de $R \wedge S$ es $sg(C_r(\bar{w})+C_s(\bar{w}))$.

(iii) la función característica de $R \vee S$ es $C_r(\bar{w})\cdot C_s(\bar{w})$

Ahora bien, supóngase que la función $f(\bar{w},z)$ es recursiva. Cuantificando se define la relacion $Q\bar{w}$ como sigue: $\bar{w} \in Q \Leftrightarrow \exists z(f(\bar{w},z)=0)$. Aunque la función f es calculable para valores arbitrarios de $\bar{w}$ y z, ello no es suficiente para decidir si $\bar{w} \in Q$ o si $\bar{w} \notin Q$ para cualquier $\bar{w}$. Por ejemplo, si para una $\bar{w}$ dada no existe una z tal que $f(\bar{w},z)=0$, el hecho de que $\bar{w} \notin Q$ no se podrá determinar mediante un cálculo. El problema radica en que la falsedad de que $\bar{w} \in Q$ involucra el cálculo de todos los valores del conjunto $\{ f(\bar{w},z) \mid z \in \mathbb{N} \}$ y dicho proceso es infinito. Este ejemplo muestra que las relaciones $\exists z R(\bar{w},z)$ y $\forall z R(\bar{w},z)$ no son necesariamente recursivas aún cuando R lo sea[1]. Esta limitación se subsana parcialmente al restringir la cuantificación a dominios acotados. Veamos la siguiente proposición.

Proposición 2.

Sea $R(\bar{w},z)$ una relación recursiva (rp) de grado n+1 y sea $f(Y)$ una función recursiva (rp) [2]. En tal caso:

(i) $S(\bar{w},Y) \equiv \exists z(z\leq f(Y) \wedge R(\bar{w},z))$ es una relación recursiva (rp) en las variables $\langle w_1,\ldots,w_n\rangle \vee \langle y_1,\ldots,y_m\rangle$.

(ii) $T(\bar{w},Y) \equiv \forall z(z\leq f(Y) \Rightarrow R(\bar{w},z))$ es una relación recursiva (rp) en las variables $\langle w_1,\ldots,w_n\rangle \vee \langle y_1,\ldots,y_m\rangle$ [3].

Demostración.

(i) $C_s(\bar{w},Y) = sg(\prod_{k=0}^{f(Y)} C_r(\bar{w},k))$. Esta función se define por substitución y es recursiva (rp) cuando f y C_r lo son.

(ii) $C_t(\bar{w},Y) = sg(\sum_{k=0}^{f(Y)} C_r(\bar{w},k))$. Mismo comentario que en (i).

(1) La relación $R(\bar{w},z) \equiv f(\bar{w},z)=0$ si es recursiva. La no recursividad de $Q(\bar{w})$ sólo puede tener por causa la cuantificación irrestricta.

(2) (rp) significa "recursiva primitiva".

(3) El grado de S o T es igual al cardinal del conjunto $\langle w_1,\ldots,w_n\rangle \vee \langle y_1,\ldots,y_m\rangle$ y es menor que n+m cuando la intersección de tales conjuntos es no vacía.

V.5 Definición de Función Recursiva Primitiva por casos.

A continuación mencionamos un procedimiento que es muchas veces usado para definir una función f con la ayuda de funciones $g_1,\dots,g_m$ ya conocidas y de predicados $t_1,\dots,t_m$ también ya conocidos. Tal definición tendrá un aspecto semejante a esto:

$$(\#) \qquad f(\bar{w}) = \begin{cases} g_1(\bar{w}), & \text{si} \quad t_1\bar{w} \\ \vdots & \qquad \vdots \\ g_m(\bar{w}), & \text{si} \quad t_m\bar{w}, \end{cases}$$ recordemos que $w \in \mathbb{N}^n$.

En base a esto, enunciamos el siguiente Teorema.

Teorema. Si $g_1,\dots,g_m$ son funciones recursivas primitivas y $t_1,\dots,t_m$ son predicados recursivos primitivos (mutuamente ajenos), entonces f, definida como en $(\#)$, es también una funcion recursiva primitiva.

Demostración.

Tenemos que, para toda $\bar{w}$ y para cualquier r $(r=1,\dots,m)$, $t_r\bar{w} \iff h_r(\bar{w})=0$, donde $h_1,\dots,h_m$ son funciones recursivas primitivas. Entonces si escribimos a f como

$f(\bar{w}) = g_1(\bar{w})\cdot\overline{sg}(h_1(\bar{w})) + \cdots + g_m(\bar{w})\cdot\overline{sg}(h_m(\bar{w}))$, tenemos que: si t_r es el único predicado que es válido para $\bar{w}$, entonces $h_r(\bar{w})=0$ y los otros $h_i(\bar{w})\neq 0$. Así, $\overline{sg}(h_r(\bar{w}))=1$ y todos los otros $\overline{sg}(h_i(\bar{w}))=0$. Por lo tanto, el lado derecho de la ecuación coincide con $g_r(\bar{w})$. Esto demuestra que f es recursiva primitiva.

Corolario.

El esquema

$$f(\bar{w}) = \begin{cases} g_1(\bar{w}), & \text{si } t_1(\bar{w}) \\ \vdots & \vdots \\ g_{m-1}(\bar{w}), & \text{si } t_{m-1}(\bar{w}) \\ g_m(\bar{w}), & \text{en cualquier otro caso} \end{cases}$$

define una función recursiva primitiva f, suponiendo que $g_1, \ldots, g_m, t_1, \ldots, t_{m-1}$ son recursivos primitivos y que $t_1, \ldots, t_{m-1}$ son mutuamente ajenos.

Esto se sigue del hecho de que la opción "en cualquier otro caso" se cumple si y sólo si $\neg t_1(\bar{w}) \wedge \cdots \wedge \neg t_{m-1}(\bar{w})$ y esto define (según la proposición 1 de la sección anterior) un predicado recursivo primitivo.

V.6 El operador μ.

El operador μ no acotado.

Sea P un predicado de $n+1$ argumentos (los cuales son números naturales) con $n \geq 0$. Si para $\bar{w}$ existe un y tal que $P\bar{w}y$, entonces para este $\bar{w}$ existe un *mínimo* y tal que $P\bar{w}y$. Denotamos este y mínimo, el cual depende de $\bar{w}$, por $\mu y P\bar{w}y$. Si, por otro lado, no existe un y para $\bar{w}$ tal que $P\bar{w}y$, entonces definimos $\mu y P\bar{w}y = 0$. Así, $\mu y P\bar{w}y$ está definido de manera única para cualquier predicado P. μ (así definido) es llamado el operador μ no-acotado. Con la ayuda de μ podemos asociar a cada predicado P de $n+1$ argumentos una función n-aria, definida como $f(\bar{w}) = \mu y P\bar{w}y$.

El operador μ no-acotado no siempre lleva de relaciones recursivas a funciones recursivas. Como en el caso de la cuantificación existencial, cuando no existe y para $\bar{w}$ tal que $P\bar{w}y$, se está ante un proceso infinito de cálculo para determinar el valor de la función. Es por ello que se introduce el operador μ en su forma acotada.

Definición.

Decimos que un predicado P, de $n+1$ argumentos, es *regular* si para cualquier $\bar{w}$ existe un y tal que $P\bar{w}y$.

El operador μ acotado.

Definición.

$$\overset{y}{\underset{z=0}{\mu}} P\bar{w}z = \begin{cases} \text{El mínimo } z \text{ entre } 0 \text{ e } y \text{ (incluyendo } 0 \text{ e } y\text{)} \\ \text{para el cual } P\bar{w}z\text{. si tal } z \text{ existe.} \\ 0 \text{, si tal } z \text{ no existe.} \end{cases}$$

Debemos considerar que al aplicar el operador μ acotado a un predicado P de $n+1$ argumentos podremos definir una función también de $n+1$ argumentos, cuyo valor depende del límite superior y.

Veamos el siguiente teorema.

Teorema. Sea P un predicado recursivo primitivo, y sea $f(\bar{w},y) = \overset{y}{\underset{z=0}{\mu}} P\bar{w}z$, entonces f es una función recursiva primitiva.

Demostración. En primer lugar verificamos las dos ecuaciones:

$$f(\bar{w},0) = 0,$$

$$f(\bar{w},S(y)) = \begin{cases} f(\bar{w},y)\text{, si existe una } z \text{ entre } 0 \text{ e } y \text{ (incluyendo } 0 \text{ e } y\text{) tal que } P\bar{w}z. \\ S(y)\text{, si el primer caso no se cumple, pero } P\bar{w}S(y), \\ 0\text{, en cualquier otro caso.} \end{cases}$$

Ahora, introducimos la función h por la definición:

$$h(\bar{w},y,t) = \begin{cases} t\text{, si existe una } z \text{ entre } 0 \text{ e } y \text{ (incluyendo } 0 \text{ e } y\text{) tal que } P\bar{w}z, \\ S(y)\text{, si el primer caso no se cumple, pero } P\bar{w}S(y). \\ 0\text{, en cualquier otro caso.} \end{cases}$$

Podemos ver, por el corolario de la sección anterior, que h es recursiva primitiva. Ahora bien, la recursividad primitiva de f es evidente, puesto que las dos ecuaciones establecidas antes pueden ser escritas en la forma:

$$f(\bar{w},0) = 0,$$

$$f(\bar{w},S(y)) = h(\bar{w},y,f(\bar{w},y)).$$

A continuación mencionamos dos ejemplos de funciones recursivas primitivas, los cuales utilizaremos en las secciones VI.3 y VI.4 .

1.- La función *número primo* $p(n)$ o p_n, la cual determina al n-ésimo número primo (es decir: $p(0)=2$, $p(1)=3$, $p(2)=5,\ldots$). Se define como

$$p(0) = 2,$$

$$p(S(n)) = \mu_{z=0}^{p(n)!+1} (Pr(z) \wedge p(n) < z).$$

en la determinacion del límite superior del operador μ se ha hecho uso del hecho de que siempre existe un número primo entre $p(n)$ y $p(n)!+1$ para toda $n \in \mathbb{N}$.

2.- La función *exponente* $\exp(n,x)$, la cual determina el exponente del número primo $p(n)$ en la descomposición prima del número x, podemos definirla (tomando en cuenta que para $x \neq 0$ el número $\exp(n,x)$ es siempre menor que x) por medio de la fórmula [1]:

$$\exp(n,x) = \mu_{z=0}^{x} \neg\, p(n)^{z+1} \mid x\,.$$

Definimos, además, $\exp(n,0) = 0$.

(1) en la sección VI.3, que es en la que se utiliza esta función, x es siempre distinta de cero.

V.7 Las Funciones σ

En el siguiente capítulo tendremos la necesidad de caracterizar parejas de números y tercias de números por números. Este es un caso de numeración de Gödel. Iniciamos con las parejas de números.

Cualquier número natural $z \geq 1$ puede ser escrito en la forma $z = 2^x(2y + 1)$, donde x e y estan determinados de manera única. En consecuencia, cualquier número $z \geq 0$ puede ser escrito en la forma $z = 2^x(2y + 1) \dot{-} 1$, donde x e y estan determinados de manera única para cada z. Ahora bien, si asociamos a cada pareja de números un número de Gödel por medio de la función

$$\sigma_2(x,y) = 2^x(2y + 1) \dot{-} 1,$$

entonces tendremos un mapeo uno a uno entre las parejas de números naturales y los números naturales. Las funciones inversas estan dadas por

$$\sigma_{21}(z) = \exp(0, z+1)$$

$$\sigma_{22}(z) = \frac{\dfrac{z+1}{2^{\exp(0,z+1)}} \dot{-} 1}{2}$$

$\sigma_2, \sigma_{21}, \sigma_{22}$ son funciones recursivas primitivas. $\sigma_{21}(z)$ y $\sigma_{22}(z)$ son la primera y la segunda componente respectivamente de la pareja de números cuyo número de Gödel es z. De manera que:

$$\sigma_{21}(\sigma_2(x,y)) = x$$

$$\sigma_{22}(\sigma_2(x,y)) = y$$

$$\sigma_2(\sigma_{21}(z), \sigma_{22}(z)) = z.$$

Ahora bien, con la ayuda de σ_2, σ_{21}, σ_{22} podemos obtener los mapeos σ_3, σ_4,... de tercias, cuartetos,... de números naturales, junto con sus correspondientes mapeos inversos. Pero debido a que para nuestro interés sólo es importante establecer hasta σ_3, escribimos este caso:

$$\sigma_3(x,y,w) = \sigma_2(\sigma_2(x,y),w) = z$$

$$\sigma_{31}(z) = \sigma_{21}(\sigma_{21}(z)) = x$$

$$\sigma_{32}(z) = \sigma_{22}(\sigma_{21}(z)) = y$$

$$\sigma_{33}(z) = \sigma_{22}(z) = w.$$

todas estas funciones son recursivas primitivas y se utilizarán en las secciones VI.3 y VI.4 .

Ahora bien, con la ayuda de σ_2, σ_{21}, σ_{22} podemos obtener los mapeos σ_3, σ_4,... de tercias, cuartetos.... de números naturales, junto con sus correspondientes mapeos inversos. Pero debido a que para nuestro interés sólo es importante establecer hasta σ_3, escribimos este caso:

$$\sigma_3(x,y,w) = \sigma_2(\sigma_2(x,y),w) = z$$

$$\sigma_{31}(z) = \sigma_{21}(\sigma_{21}(z)) = x$$

$$\sigma_{32}(z) = \sigma_{22}(\sigma_{21}(z)) = y$$

$$\sigma_{33}(z) = \sigma_{22}(z) = w.$$

todas estas funciones son recursivas primitivas y se utilizarán en las secciones VI.3 y VI.4 .

C A P I T U L O   V I

EQUIVALENCIA ENTRE RECURSIVIDAD Y TURING-CALCULABILIDAD

En este capítulo se va ha demostrar que la clase de las funciones aritméticas recursivas coincide con la clase de las funciones aritméticas que son turing-calculables. Por esta razón, en la demostración se considera que las funciones mencionadas están definidas para toda n-ada de números naturales y que sus valores son también números naturales. Dicho lo anterior, pasamos a enunciar los dos teoremas más importantes de este capítulo, los cuales demostraremos más adelante.

TEOREMA I. *Toda Función Aritmética que es Recursiva es Turing-Calculable.*

TEOREMA II. *Toda Función Aritmética que es Turing-Calculable es Recursiva.*

Se demostrarán estos teoremas en forma rigurosa sin hacer uso del concepto intuitivo de Calculabilidad.

De los teoremas anteriores podamos deducir los siguientes corolarios.

Corolario 1 : *Todo Predicado Recursivo es Turing-Decidible.*

Corolario 2 : *Todo Predicado Turing-Decidible es Recursivo.*

Demostración del Corolario 1.

Sea P un Predicado Recursivo. Por definición, existe una función recursiva f tal que, para cada $\bar{w}$, $f(\bar{w}) = 0$ si y sólo si $P\bar{w}$. Ahora bien, como f es turing-calculable, por el teorema I, sea M la máquina de Turing que calcula a f. Entonces la máquina MI^2 aplicada sobre un cuadro arbitrario de la cinta, en la cual sólo esta escrito el argumento $\bar{w}$, se detiene, después de un número finito de pasos, sobre $*$ si P es válido para $\bar{w}$ (es decir, si $f(\bar{w}) = 0$) o sobre $/$ si P no es válido para $\bar{w}$ (es decir, si $f(\bar{w}) > 0$). Recordemos que los números naturales estan representados por secuencias de trazos, según la descripción vista en la sección I.4 . Esto demuestra que P es Turing-Decidible.

Demostración del Corolario 2.

Sea P un Predicado Turing-Decidible y sea M una máquina de Turing que decide a P. De modo que si M es aplicada sobre un cuadro arbitrario de la semicinta, en la cual sólo esta escrito el argumento $\bar{w}$, entonces se detiene sobre $*$ si P es válido para $\bar{w}$ o sobre $/$ en caso contrario. Entonces la máquina

$$M \begin{array}{l} \xrightarrow{\qquad 0 \qquad} D/D* \\ \xrightarrow[\qquad 1 \qquad]{} *D/D/D* \end{array}$$

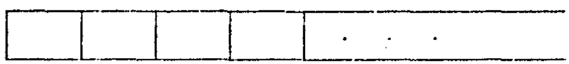
calcula una función f tal que $f(\bar{w}) = 0$ si y sólo si P es válido para $\bar{w}$. De manera que f es Turing-Calculable y, por el Teorema I.., f es Recursiva [illegible] P es Recursivo.

VI.1 Turing-Calculabilidad Estandar.

El Teorema I se demuestra en forma modificada. Para ello se requiere del concepto de Turing-Calculabilidad en forma estandar. Este concepto difiere del concepto de Turing-Calculabilidad ya introducido. En primer lugar, trataremos con funciones cuyos valores y argumentos son números naturales (es decir, secuencias de trazos), mientras que para Turing-Calculabilidad admitimos palabras (no vacías) de cualquier alfabeto como valores y argumentos. El alfabeto que usaremos es { / }. Además, tenemos otras tres diferencias importantes:

1.- El cuadro inspeccionado al iniciar el cálculo esta definido, lo cual es una simplificación.

2.- Los cálculos se realizan sobre una SEMICINTA, la cual consta de un cuadro inicial y se extiende infinitamente hacia la derecha. Su aspecto es el siguiente:



3.- Una lista de condiciones es dada. Mencionamos éstas a continuación. Tales condiciones tienen como ventaja, como veremos más tarde, que podemos construir con facilidad máquinas (que calculan funciones más complicadas) desde máquinas que satisfacen estas condiciones:

(i) aceptaremos que a la izquierda de los argumentos dados (con el respectivo espacio entre ellos) la semicinta puede tener impresas otras cosas. No obstante, a la derecha de los argumentos la semicinta esta vacía. En el caso de una función de cero argumentos, puede haber impresas algunas cosas a la izquierda del cuadro inspeccionado al momento de iniciar el cálculo de dicha función. Ahora bien, para una formulación más conveniente de la definición es importante mencionar el siguiente concepto:

Lista Argumental (de una función n-aria) : es el fragmento $*\rho_1*\rho_2\cdots*\rho_n$ formado por los argumentos. De manera que el primer cuadro de la lista argumental está vacío y el último cuadro tiene al último símbolo de ρ_n escrito en él. Esto es válido para $n \geq 1$. Una lista argumental de una función de cero argumentos no contendrá cuadro alguno.

(ii) aceptaremos también que, durante el cálculo del valor de la función, sólo la lista argumental y la semicinta a su derecha son escudriñados.

Definición .

Sea **f** una función n-aria ($n\geq1$), **f** es *Turing-Calculable en forma estandar* si existe una máquina de Turing **M** sobre el alfabeto { / } tal que, para toda lista argumental, si aplicamos **M** sobre el primer cuadro vacío a la derecha de la lista argumental, entonces M deja de operar después de un número finito de pasos y al final del cálculo tenemos que :

(i) los argumentos estan en el mismo lugar que al principio,

(ii) el valor de la función empieza en el segundo cuadro a la derecha de la lista argumental, de manera que hay un solo espacio entre el valor de la función y los argumentos.

(iii) **M** está sobre el cuadro inmediato a la derecha del último trazo del valor de la función, y

(iv) la semicinta a la derecha del valor de la función está vacía.

Modo de Operación

$$*a_1*a_2\cdots*a_n\underline{*}\cdots \Rightarrow *a_1*a_2\cdots*a_n*f(a_1,\ldots,a_n)\underline{*}\cdots$$

En el caso de las funciones de cero argumentos modificamos la definición de Calculabilidad estandar del modo siguiente :

Una función f de cero argumentos es llamada *Turing-Calculable en forma estandar* si existe una máquina de Turing M sobre el alfabeto { / } tal que si aplicamos M sobre un cuadro arbitrario Q (vacío) y la semicinta a la derecha de tal cuadro se encuentra vacía, entonces M deja de operar después de un número finito de pasos y al final del cálculo tenemos que :

(i) el valor de la función empieza en el primer cuadro a la derecha de Q.

(ii) M está sobre el cuadro inmediato a la derecha del último trazo del valor de la función.

(iii) la semicinta a la derecha del valor de la función está vacía.

Modo de Operación

$$\underset{\substack{\uparrow \\ \text{cuadro } Q}}{\underline{\times}}\cdots \quad \Rightarrow \quad \underset{\substack{\uparrow \\ \text{cuadro } Q}}{\times} f \underline{\times}\cdots$$

VI.2 La Turing-Calculabilidad de las funciones Recursivas.

Toda función Turing-Calculable en forma estandar es también Turing-Calculable (a consecuencia del Teorema 1 de la sección IV.7). Lo sorprendente es que el recíproco de este enunciado también es cierto. Veamos :

Teorema A . Toda Función Recursiva es Turing-Calculable en forma estandar.

Ahora bien, el teorema A implica el teorema I . Y por otro lado, el teorema A junto con el teorema II implican que

Toda función aritmética que es turing-calculable
es turing-calculable en forma estandar.

De manera que para encontrar el valor de funciones calculables, podemos hacer uso de una máquina de Turing que utiliza un solo símbolo (aparte del símbolo vacío).

Procedemos con la prueba del teorema A.

La demostración es por inducción sobre el nivel de Recursividad de la función.

Demostración (del Teorema A).

(i) Veamos primero el caso en que el nivel de recursividad de la función es cero, es decir cuando f es una función inicial.

a) La función Sucesor es Turing-Calculable en forma estandar por la máquina C/D.

b) La función $I_{n,k}$ $(1 \leq k \leq n)$ es Turing-Calculable en forma estandar por la máquina C_{n+1-k} .

c) La función $I_{0,0}$ es Turing-Calculable en forma estandar por la máquina D/D.

Denotaremos, de aquí en adelante, los argumentos por X. Es decir $X = x_1 * x_2 \cdots * x_n$.

(ii) Si f es una función de n-argumentos $(n \geq 0)$ y fue obtenida por composición mediante la ecuación

$$f(X) = g(h_1(X),\ldots,h_r(X)) \quad , \text{con } r \geq 1$$

Sean $M, M_1, \ldots, M_r$ máquinas de Turing que calculan en forma estandar a las funciones $g, h_1, \ldots, h_r$ respectivamente. Entonces

f es Turing-Calculable en forma estandar por:

$$C_n^n[D(1_d B_t)^n L_d]^2 M_1 C_{n+1}^n M_2 \cdots C_{n+1}^n M_r C_{r+(r-1)n} C_{r+(r-2)n} \cdots C_r M B_r M^0 \quad \text{si} \quad n \geq 1.$$

o por $\quad D^2 M_1 M_2 \cdots M_r M B_r M^0 \quad$ si $\quad n=0.$

La demostración se hará para $n\geq 1$, el caso $n=0$ es fácil de comprobar.

Demostración.

Al iniciar el cálculo, tenemos

$$* X \underline{*} \cdots .$$

En Primer lugar, la máquina C_n^n copia los n argumentos, es decir, al aplicarla se obtiene

$$* X * X \underline{*} \cdots .$$

Después con la ayuda de la máquina $[D(1_d B_t)^n L_d]^2$ recorremos la copia de los argumentos dos lugares a la derecha. Esto es

$$* X * * * X \underline{*} \cdots .$$

Ahora utilizamos M_1 para calcular $h_1(X)$, o sea

$$* X * * * X * h_1 \underline{*} \cdots ,$$

donde h_1 es abreviatura de $h_1(X)$.

Ahora bien, para calcular $h_2(X)$ necesitamos tener los argumentos en el extremo derecho. Para lo cual utilizamos la máquina C_{n+1}^n : $\quad * X * * * X * h_1 * X \underline{*} \cdots .$

A continuación, calculamos h_2 por medio de M_2 :

$$* X * * * X h_1 * X * h_2 \underline{*} \cdots ,$$

donde h_2 es abreviatura de $h_2(X)$.

Del mismo modo, por medio de $C_{n+1}^n M_3 \cdots C_{n+1}^n M_r$, podemos ir acomodando los argumentos e ir calculando $h_3, h_4, \ldots, h_r$:

$$* X * * * X * h_1 * X * h_2 \cdots * X * h_r \underline{*} \cdots .$$

Para poder calcular $g(h_1,\ldots,h_r)$ debemos tener los valores de $h_1,\ldots,h_r$ a nuestra disposición en el extremo derecho. Para esto h_i es copiado por la máquina $C_{r+(r-1)n}$. Por lo tanto, en la semicinta, tenemos

$$* \; X \; * \; * \; * \; X \; * \; h_1 \; * \; X \; * \; h_2 \cdots * \; X \; * \; h_r \; * \; h_1 \; * \; h_2 \cdots h_r \; \underline{*} \cdots .$$

En seguida $f(X) = g(h_1(X),\ldots,h_r(X))$ se calcula con la ayuda de la máquina M :

$$* \; X \; * \; * \; * \; X \; * \; h_1 \; * \; X \; * \; h_2 \cdots * \; X \; * \; h_r \; * \; h_1 \; * \; h_2 \cdots h_r \; * \; f \; \underline{*} \cdots .$$

Para terminar, se aplican la máquina B_r para borrar los cálculos intermedios y la máquina M^0 para indicar que el cálculo terminado :

$$* \; X \; * \; f \; \underline{*} \cdots .$$

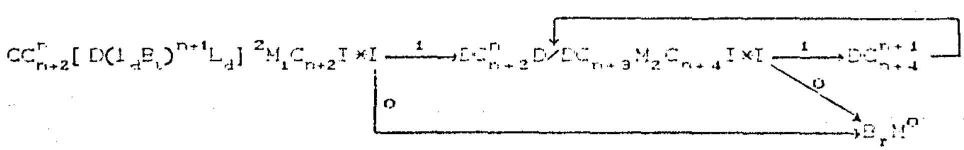
De este modo, queda demostrada la Turing-Calculabilidad en forma estandar de f.

(iii) Si f es una función de $(n+1)$ argumentos, con $n \geq 0$, y fue obtenida por recursión mediante las ecuaciones:

$$f(X,0) = g(X)$$

$$f(X,S(y)) = h(X,y,f(X,y)) .$$

Sean M_1 y M_2 Máquinas de Turing que calculan en forma estandar a las funciones g y h respectivamente. Entonces f es Turing-Calculable en forma estandar por la máquina :



En la demostración se usa la abreviatura f_y en lugar de $F(X,y)$.

Demostración.

Al iniciar el cálculo, se tiene

$$* \, X * y \, \underline{*} \cdots .$$

Ahora aplicando la máquina CC^{n}_{n+2} tenemos

$$* \, X * y * y * X \, \underline{*} \cdots .$$

A continuación, con la ayuda de la máquina $[D(I_d B_i)^{n+1} L_d]^2$, tenemos

$$* \, X * y * * * y * X \, \underline{*} \cdots .$$

Ahora se obtiene el valor f_o de la función aplicando M_1:

$$* \, X * y * * * y * X * f_o \, \underline{*} \cdots .$$

En seguida, copiamos "y", borramos el último trazo de dicha copia, y retrocedemos un cuadro por medio de $C_{n+2} I * I$.

Ahora bien, si la máquina está sobre un cuadro vacío, entonces $y = 0$, y con f_o obtuvimos el valor de la función. Por lo cual, para terminar, aplicamos la máquina B_r para borrar los cálculos intermedios y la máquina M^o para indicar que el cálculo ha terminado. En la semicinta, tenemos:

$$* \, X * y * f_o \, \underline{*} \cdots .$$

No obstante, si existe un trazo en el cuadro inspeccionado, entonces el cálculo no ha finalizado. En tal caso, nos movemos un cuadro a la derecha (D), quedando en seguida de $y-1$:

$$* \, X * y * * * y * X * f_o * y-1 \, \underline{*} \cdots .$$

Ahora necesitamos calcular $f_1, f_2, \ldots, f_y$, sucesivamente, con la ayuda del proceso de inducción. Para esto primero copiamos X, a continuación escribimos un CERO y finalmente copiamos f_o. Después de esto, la máquina estará a la derecha de $X * 0 * f_o$, es decir, en seguida de los argumentos necesarios para calcular f_1.

Todo esto lo obtenemos por medio de

$$C^{n}_{n+2}D/DC_{n+3}M_2 \qquad (*)$$

después de lo cual el aspecto de la semicinta es

$$* X * y * * * y * X * f_0 * y-1 * X * 0 * f_0 * f_1 \underline{*} \cdots .$$

A continuación, copiamos $y-1$, borramos el último trazo de dicha copia y retrocedemos un cuadro con la ayuda de

$$C_{n+4} I * I .$$

Ahora bien, si la máquina esta sobre un cuadro vacío, entonces $y-1 = 0$ (es decir $y = 1$). Y con $f_y = f_1$ obtuvimos el valor de la función. Por lo que, para terminar, aplicamos B_r para borrar los cálculos intermedios y M_0 para indicar el fin del cálculo

$$* X * y * f_1 \underline{*} \cdots .$$

Sin embargo, si la máquina está sobre un trazo, entonces $y-1 \neq 0$. Para continuar con el proceso, obtenemos los argumentos X, 0 con la ayuda de la máquina DC^{n+1}_{n+4}. El argumento 0 es incrementado en 1 por $/D$, y f_1 es copiado por medio de C_{n+3}, después de lo cual el último proceso considerado es repetido. Es decir, repetimos desde donde se encuentra $/D$ en la expresión $(*)$. De modo que la semicinta va tomando los siguientes aspectos:

$$* X * y * * * y * X * f_0 * y-1 * X * 0 * f_0 * f_1 * y-2 * X * 1 * f_1 * f_2 \underline{*} \cdots$$

hasta obtener finalmente

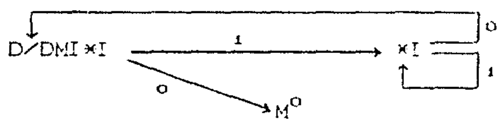
$$* X * y * * * y * \cdots\cdots\cdots\cdots f_{y-1} * y-y * X * y-1 * f_{y-1} * f_y \underline{*} \cdots$$

Terminándose el cálculo aplicando las máquinas B_r y M^0 :

$$* X * y * f_y \underline{*} \cdots .$$

(iv) Sea g una función de $(n+1)$ argumentos, con $n \geq 0$, tal que para toda X existe y para la cual $g(X,y) = 0$. Y sea f la función definida por la ecuación : $f(X) = \mu\, y(g(X,y) = 0)$.

Si M es una máquina de Turing que calcula a g en forma estandar, entonces f es Turing-Calculable en forma estandar por la máquina



Al inicio del cálculo la máquina se encuentra en seguida de los argumentos X, estando la semicinta vacía a la derecha de X.

Demostración.

1) La máquina se mueve un cuadro a la derecha (D).

2) La máquina escribe un trazo (/) y se mueve un cuadro a la derecha (D). Ahora, tenemos los argumentos X, y para calcular la función g (en un principio $y = 0$). La máquina calcula $g(X,y)$ a través de (M). A continuación, borramos el último trazo del valor de $g(X,y)$ por medio de ($I*$) y vamos (con la ayuda de I) un cuadro hacia la izquierda. Dependiendo de si la máquina está sobre un cuadro marcado o sobre un cuadro vacío, vamos al paso 3 o al paso 4 respectivamente.

3) Si $g(X,y) \neq 0$, entonces debemos probar con el próximo número y . Con la ayuda de $*I$ ^{1} la máquina borra el resto de $g(X,y)$. Y repitiendo desde el paso número 2, el procedimiento se realiza ahora para $y+1$ en lugar de y.

4) Si $g(X,y) = 0$ entonces el mínimo y para el cual $g(X,y)$ es igual a cero se encuentra inmediatamente a la izquierda del cuadro inspeccionado en este momento. Además sabemos que la semicinta, a la derecha del valor de la función, está vacía. Por lo tanto, el cálculo ha terminado.

Queda demostrado que Toda Función Recursiva es Turing-Calculable en Forma Estandar. Cabe señalar que la demostración es constructiva en el sentido siguiente: Dada una función recursiva, junto con la sucesión de funciones que la definen, se puede construir una Máquina de Turing que la calcula en forma estandar [1].

(1) En el PROGRAMA hay un Constructor de Máquinas de Turing que Calculan Funciones Recursivas. Para mayor información consultar, en el apéndice C, la parte de "Ligador de Tablas de una Digráfica".

VI.3 Numeración de Gödel de Máquinas de Turing.

A continuación llevaremos a cabo distintas numeraciones de Gödel que iremos utilizando conforme avancemos. En primer lugar, vamos a denotar los cuadros de la cinta por números naturales y caracterizaremos sobre esta base las configuraciones de una Máquina de Turing por números. Además, construiremos un mapeo uno-uno entre Máquinas de Turing arbitrarias y algunos números. Sobre estas bases introduciremos funciones por medio de las cuales podremos determinar el número de la configuración siguiente desde el número de una configuración dada.

Numeración de los cuadros de la cinta.

Seleccionamos un cuadro arbitrario de la cinta de una Máquina de Turing y le damos el número 0. Numeramos los otros cuadros de acuerdo al siguiente esquema:

. . .	7	5	3	1	0	2	4	6	8	. . .

Hablaremos, según esta numeración, del cuadro x. A la derecha del cuadro x esta el cuadro $D(x)$ y a la izquierda el cuadro $I(x)$. Si Mx significa que x es par y Nx significa que x es impar, tenemos:

$$D(x) = \begin{cases} x + 2 \quad , & \text{si} \quad Mx \\ 0 \quad , & \text{si} \quad x = 1 \\ x - 2 \quad , & \text{si} \quad Nx \wedge x \neq 1 \quad . \end{cases}$$

$$I(x) = \begin{cases} x + 2 \quad , & \text{si } Nx \\ 1 \quad , & \text{si } x = 0 \\ x \dot{-} 2 \quad , & \text{si } \bar{N}x \wedge x \neq 0 \quad . \end{cases}$$

$D(x)$ e $I(x)$ son funciones recursivas primitivas.

xDy querrá decir que el cuadro x está a la derecha del cuadro y. Tenemos

$$xDy \iff (\bar{N}x \wedge \bar{N}y \wedge x > y) \vee (\bar{N}x \wedge Ny) \vee (Nx \wedge Ny \wedge x < y).$$

Esta definición muestra que D es un predicado recursivo primitivo.

Necesitamos otra función $Z(x,y)$ la cual es el número de cuadros que están situados entre x e y (incluyendo al cuadro x pero no al cuadro y) a condición de que y esté a la izquierda de x. $Z(x,y)$ es una función recursiva primitiva porque

$$Z(x,y) = \begin{cases} \dfrac{x \dot{-} y}{2} \quad , & \text{si } \bar{N}x \wedge \bar{N}y \\ \dfrac{y \dot{-} x}{2} \quad , & \text{si } Nx \wedge Ny \\ \dfrac{(x + y) - 1}{2} \quad , & \text{en cualquier otro caso.} \end{cases}$$

Caracterización de las expresiones de cinta por números.

La numeración de cuadros que acabamos de definir hace posible caracterizar una expresión de cinta por un número b de un modo sencillo. Hablaremos, en este sentido, de la *expresión de cinta* b. Supongamos que el cuadro j contiene la letra $a_{b(j)}$. Entonces definimos

$$b = \prod_{j=0}^{\infty} p_j^{b(j)} \quad .$$

Nota: Hay dos posibilidades: $a_{b(j)} = a_1$ (el trazo) o $a_{b(j)} = a_0$ (el vacío). En consecuencia el número b es un producto finito de números primos sin repetición. Puesto que todos los exponentes son 0 o 1.

Recordemos que un cuadro vacío tiene la letra a_0 impresa en él, de modo que los cuadros vacíos proporcionan el factor 1 al producto, el cual es por tanto infinito sólo en un sentido formal. $b = 1$ denota una cinta vacía. Si la cinta tiene la expresión b, entonces el cuadro j tiene la letra $a_{\exp(j,b)}$ escrita en él. Recordar que $\exp(j,b)$ denota al exponente del número primo $p(j)$ en la descomposición prima del numero b.

Supongamos que el cálculo de una función ha terminado. Entonces encontramos sobre la cinta la expresión b. Sea el cuadro α el último cuadro inspeccionado (vacío). Según la sección IV.4 la palabra que representa el valor de la función termina justo antes del cuadro α. El valor w de la función es igual al número de trazos que forman esta palabra, disminuido en uno. w esta determinado por α y b. Queremos describir una función $W_0(\alpha,b)$ tal que $w = W_0(\alpha,b)$. Para esto primero describimos los cuadros que determinan el extremo izquierdo $E_i(\alpha,b)$ y el extremo derecho $E_d(\alpha,b)$ de la palabra en cuestión.

El extremo derecho es claramente

$$E_d(\alpha,b) = I(\alpha) .$$

El extremo izquierdo está caracterizado en forma única por dos condiciones:

(1) El cuadro situado a su izquierda está vacío, y

(2) Cualquier cuadro que este entre el cuadro situado a su izquierda y el cuadro α contiene al símbolo $a_1 = /$.

Estas condiciones nos dan

$$E_i(\alpha,b) = \mu x[\, \mathrm{epx}(I(x),b)=0 \;\wedge\; \forall y(yDI(x) \wedge \alpha Dy \Rightarrow \exp(y,b)=1)] .$$

E_d y E_i son funciones recursivas primitivas. Esto sólo necesitamos demostrarlo para E_i. El primer paso es dar cotas para el operador μ y para el generalizador $\forall$ que aparecen en la definición de E_i. Para encontrar x sólo necesitamos considerar aquellos números para los cuales el cuadro x está marcado. De manera que $p(x)$ debe dividir a b. Este hecho hace evidente que $x \le b$. Para y podemos escoger $\max(I(x),a)$ como cota. Veamos, necesitamos considerar y tal que $yDI(x)$ y aDy. Si $yDI(x)$ entonces y es par o $y < I(x)$.

Si $y < I(x)$, entonces $y \le \max(I(x),a)$. Por otro lado, si y es par entonces, como aDy, tenemos que $y < a$. Por lo tanto, en cualquier caso, $y \le \max(I(x),a)$. En consecuencia

$$E_i(a,b) = \mu_{x=0}^{b} [\exp(I(x),b)=0 \wedge \forall_{y=0}^{\max(I(x),a)} (yDI(x) \wedge aDy \Rightarrow \exp(y,b)=1)].$$

Es claro que

(*) $$W_0 = Z(E_d(a,b), E_i(a,b)).$$

Esto demuestra que W_0 es recursiva primitiva.

El Número de Gödel t de una Máquina de Turing M.

M esta dada por una tabla. M tiene $m+1$ estados $0,\ldots,m$ $(m \ge 0)$ y trabaja con los símbolos a_0, a_1. En la tercera columna estan las instrucciones y en la cuarta columna los nuevos estados.

Reemplazaremos los símbolos de la tercera columna (por números) de la siguiente manera:

$$I \;,\; D \;,\; P \;,\; a_0 \;,\; a_1$$

son reemplazados por

$$1 \;,\; 2 \;,\; 3 \;,\; 4 \;,\; 5$$

respectivamente.

Después de esta alteración tenemos que los elementos A_{ij} (de la matriz) son números para $i = 1,\ldots,2(m+1)$ y $j = 3,4$. Esto lo caracterizamos por el número

$$t = \rho_0\ \rho_1^m \prod_{i=1}^{2(m+1)} \prod_{j=3}^{4} \rho_{\sigma_2(i,j)}^{A_{ij}}$$

t es llamado El número de Godel de M.

Desde t podemos obtener la tabla de M. En primer lugar, tenemos que $\sigma_2(i,j) > 1$ para $j = 3$ o $j = 4$. Además, sabemos que $n = 1$ y que $m = exp(1,t)$. Por otro lado, tenemos que, para $i = 1,\ldots,2(m+1)$ y $j = 3,4$,

$$A_{ij} = exp(\sigma_2(i,j),t).$$

De modo que, los 2 renglones de la tabla de M que pertenecen al estado c ($c=0,\ldots,m$) tienen un aspecto semejante a esto

c	a_0	$exp(\sigma_2(2c+0+1,3),t)$	$exp(\sigma_2(2c+0+1,4),t)$
c	a_1	$exp(\sigma_2(2c+1+1,3),t)$	$exp(\sigma_2(2c+1+1,4),t)$.

Si además definimos la abreviatura

$$h(p,q,c,t) = exp(\sigma_2(2c+q+1,p),t)$$

entonces el renglón de la tabla de M que empieza con ca_q será

(* *) $\qquad$ c $\qquad a_q \qquad h(3,q,c,t) \qquad h(4,q,c,t)$.

Podemos observar que la función h es recursiva primitiva.

. El procedimiento siguiente proporciona una manera de determinar si un número arbitrario t es o no el número de Godel de un Máquina de Turing. Sabemos que $n = 1$. Ahora calculamos el número m (como acabamos de ver). En seguida, usando (**), producimos una matríz de $2(m+1)$ renglones y de cuatro columnas. Ahora, debemos comprobar que las siguientes condiciones se cumplan:

(1) $\exp(0,t) = 1$,

(2) $1 \leq h(3,q,c,t) \leq 5$,

(3) $h(4,q,c,t) \leq m$.

Si estas condiciones no se cumplen, entonces t (claramente) no es el número de Godel de una máquina de Turing. Por otro lado, si las condiciones se cumplen, entonces la matríz producida nos da una Máquina de Turing. Para esta máquina debemos ahora calcular su número de Godel t_0. Ahora bien, es claro que t es el número de Godel de un máquina de Turing si y sólo si $t = t_0$.

Comentario. Es necesario checar que $t = t_0$ por la siguiente razón: Si multiplicamos el número de Godel t_0 de una máquina de Turing M_0 por un número primo que no sea factor de t_0, entonces obtenemos un número t que (ciertamente) no es el número de Gödel de una máquina de Turing. Por otro lado, el procedimiento descrito arriba, si se aplica a t producirá la tabla de M_0.

Las Funciones A, B, C.

Sea t el número de Gödel de una máquina de Turing. Consideremos la configuración (según la sección IV.3) denotada por el cuadro inspeccionado a, la expresión de cinta b y el estado c. Si la máquina no dejó de operar en esta configuración entonces obtenemos una configuración consecutiva denotada por un nuevo cuadro inspeccionado, el cual está determinado en forma única por t,a,b,c y puede ser escrito en la forma $A(t,a,b,c)$, una nueva expresión de cinta $B(t,a,b,c)$, y un nuevo estado $C(t,a,b,c)$. A continuación damos la definición de las funciones A, B, C.

En un principio, el cuadro inspeccionado a contiene el símbolo $a_{\exp(a,b)}$. De manera que la línea que es decisiva para el próximo paso de la máquina es aquella que empieza con $ca_{\exp(a,b)}$, la cual según (**) es

$$(***) \quad c \quad a_{\exp(a,b)} \quad h(3,\exp(a,b),c,t) \quad h(4,\exp(a,b),c,t).$$

Esto nos proporciona el nuevo estado. Es decir

$$C(t,a,b,c) = h(4,\exp(a,b),c,t).$$

El nuevo cuadro inspeccionado está a la izquierda o a la derecha del anterior cuadro inspeccionado, si $h(3,\exp(a,b),c,t) = 1$ o 2 respectivamente. En cualquier otro caso el cuadro inspeccionado permanece sin cambio. Es decir

$$A(t,a,b,c) = \begin{cases} I(a) \ , & \text{si} \quad h(3,\exp(a,b),c,t) = 1 \\ D(a) \ , & \text{si} \quad h(3,\exp(a,b),c,t) = 2 \\ a \ , & \text{en cualquier otro caso.} \end{cases}$$

La expresión de cinta es alterada sólo si otro símbolo es escrito en el cuadro inspeccionado. La alteración será descrita multiplicando o dividiendo por una adecuada potencia de ρ_a. Tenemos

$$B(t,a,b,c) = \begin{cases} \dfrac{b \cdot \rho_a^{h(3,\exp(a,b),c,t)-4}}{\rho_a^{\exp(a,b)}} \ , & \text{si} \quad 4 \le h(3,\exp(a,b),c,t) \\ b \ , & \text{si} \quad 4 > h(3,\exp(a,b),c,t). \end{cases}$$

Las definiciones dadas demuestran que A, B, C son funciones recursivas primitivas.

Los valores $A(t,a,b,c)$, $B(t,a,b,c)$, $C(t,a,b,c)$ tienen el significado dado sólo si (a,b,c) no es una configuración terminal.

Vamos a considerar un predicado más, a saber E_0tabc. El cual, bajo la suposición de que t es el número de Godel de una máquina de Turing M, quiere decir que la configuración (a,b,c) es una configuración terminal de M. Este es el caso si y sólo si la línea de la tabla que empieza con $ca_{vab+1,b}$ tiene el símbolo P (Parar) en la tercera columna. Hemos representado este símbolo por el número 3. Por lo tanto, según (***), tenemos que $E_0tabc \Leftrightarrow h(3,exp(a,b),c,t) = 3$. De manera que E_0 es recursivo primitivo.

VI.4 La Recursividad de Las Funciones Turing-Calculables.

El Teorema II es un corolario del siguiente teorema que llamamos de Kleene:

Teorema de Kleene:

Existe una función singular recursiva primitiva U y, para cada n, un predicado recursivo primitivo T_n de grado n+2, con la siguiente propiedad: Si f es una función Turing-calculable de $\mathbb{N}^n$ en $\mathbb{N}$, entonces existe un número t tal que para cada $\bar{w} \in \mathbb{N}^n$

(i) existe $y \in \mathbb{N}$ tal que $T_n t\bar{w}y$,

y (ii) $f(\bar{w}) = U(\mu y T_n t\bar{w}y)$.

El inciso (ii) demuestra la recursividad de f. Puesto que f se puede calcular aplicando una sola vez el operador μ a un predicado regular, seguida la aplicación de la función U. Nótese que para una n-ada, todas las funciones Turing-Calculables de ese grado se generan haciendo variar a t.

Del Teorema I y del Teorema de Kleene, se obtiene el

Teorema de la Forma Normal de Kleene:

Existe una función recursiva primitiva U y, para cada $n \in \mathbb{N}$, un predicado recursivo primitivo T_n de grado $n+2$, con la siguiente propiedad: Si f es una función recursiva de $\mathbb{N}^n$ en $\mathbb{N}$, entonces hay un número t tal que para cada $\vec{w} \in \mathbb{N}$

(i) existe una $y \in \mathbb{N}$ tal que $T_n t\vec{w}y$,

y (ii) $f(\vec{w}) = U(\mu y T_n t\vec{w}y)$.

PROBAREMOS AHORA EL TEOREMA DE KLEENE.

1. Números de Godel de Configuraciones, y funciones y predicados asociados con ellas.

Sea M una máquina de Turing arbitraria. Una configuración de M está dada por una terna (a,b,c). Caracterizamos esta configuración unívocamente por el número $\sigma_3(a,b,c)$ el cual llamamos *el número de Gödel de la configuración* (a,b,c). Hablaremos, en este sentido, de la configuración k. Naturalmente,

$$a = \sigma_{31}(k), \quad b = \sigma_{32}(k), \quad c = \sigma_{33}(k).$$

Suponiendo que t es el número de Godel de una máquina de Turing, Etk significa que k es el número de Godel de una configuración terminal de la máquina de Turing cuyo número es t. Según la sección anterior, tenemos que

$$Etk \iff E_0 t\sigma_{31}(k)\sigma_{32}(k)\sigma_{33}(k).$$

Ahora bien, suponiendo que k tiene una configuración consecutiva. El número de Godel de tal configuración está dado por la función $F(t,k)$ la cual, según la sección anterior, está definida como

$$F(t,k) = \sigma_3(A(t,\sigma_{31}(k),\sigma_{32}(k),\sigma_{33}(k)), B(t,\sigma_{31}(k),\sigma_{32}(k),\sigma_{33}(k)), C(t,\sigma_{31}(k),\sigma_{32}(k),\sigma_{33}(k))).$$

Finalmente, consideremos una configuración k en la cual el cuadro inspeccionado está vacío y está en seguida de una secuencia de trazos (los cuales representan el valor de una función). Dicho valor está dado por $W(k) = W_0(z_{31}(k), z_{32}(k))$.

Es claro que E, F y W son recursivas primitivas.

2. La función $K(t, \bar{w}, z)$.

Partimos de una Máquina de Turing arbitraria M con número de Gödel t. Además, sea $\bar{w}$ una n-ada arbitraria de argumentos. Escribimos $\bar{w}$ sobre la cinta vacía. Elegimos como el cuadro con el número cero al primer cuadro de la lista argumental (un cuadro arbitrario si $n = 0$). A continuación obtenemos una secuencia de configuraciones las cuales tal vez lleguen a un término. Los números de estas configuraciones dan una secuencia $K(t, \bar{w}, z)$, $z = 0, 1, 2, \ldots$. Si la secuencia de configuraciones termina después de z_0 pasos, entonces $K(t, \bar{w}, z)$ esta determinado sólo para $z \leq z_0$. En tal caso definiremos $K(t, \bar{w}, z)$ para $z > z_0$ de la siguiente manera $K(t, \bar{w}, z) = K(t, \bar{w}, z_0)$.

Observamos dos propiedades de K.

(2.1) Si $K(t, \bar{w}, z) = K(t, \bar{w}, z')$, entonces

$$K(t, \bar{w}, z) = K(t, \bar{w}, z') = K(t, \bar{w}, z'') = \cdots .$$

Para probar esto consideramos dos casos:

(a) $K(t, \bar{w}, z)$ no es una configuración terminal. Entonces, puesto que $K(t, \bar{w}, z) = K(t, \bar{w}, z')$, la configuración $K(t, w, z)$ es su propia configuración consecutiva. Ahora bien, debido a la unicidad de la configuración consecutiva todas las configuraciones siguientes deben coincidir con $K(t, \bar{w}, z)$.

(b) $K(t,\bar{w},z)$ es una configuración terminal. Este caso sólo puede ocurrir si M deja de operar después de un número finito de pasos, digamos z_0 pasos. Entonces ninguna configuración $K(t,\bar{w},u)$ con $u < z_0$ es una configuración terminal, además $z_0 \leq z$. Pero, según la definición de la función K tenemos que, para cualquier $z \geq z_0$, $K(t,\bar{w},z) = K(t,\bar{w},z_0)$, de donde se concluye la propiedad mencionada.

(2.2) Si M deja de operar después de z_0 pasos, entonces

$$z_0 = \mu z(K(t,\bar{w},z) = K(t,\bar{w},z')).$$

Sea $z_1 = \mu z(K(t,\bar{w},z') = K(t,\bar{w},z'))$. De la definición de K se sigue que $z_1 \leq z_0$. Si $z_1 < z_0$, entonces según (2.1) tenemos que $K(t,\bar{w},z_1) = K(t,\bar{w},z_1') = \cdots = K(t,\bar{w},z_0)$ es una configuración terminal, en contradicción al hecho de que $K(t,\bar{w},z_0)$ es la primera configuración terminal de la secuencia de configuraciones $K(t,\bar{w},z)$. Por lo tanto $z_1 = z_0$.

A continuación, definiremos por recursión a la función $K(t,\bar{w},z)$. En primer lugar, $K(t,\bar{w},0)$ es el número de la configuración inicial. Aquí, $c = 0$. La expresión de cinta inicial consiste de los argumentos de $\bar{w}$. Si notamos que todos los cuadros a la derecha de 0 tienen números pares asociados con ellos, entonces es fácil notar que la expresión de cinta inicial está dada por

$$b_0(\bar{w}) = \frac{\prod_{j=0}^{x_1+\cdots+x_n+2n} \rho(2j)}{\rho(2(x_1+\cdots+x_n+2n))\rho(2(x_1+\cdots+x_{n-1}+2(n-1)))\cdots\rho(2(x_1+2))\rho(0)}$$

El cuadro inspeccionado de la configuración inicial, es el que se encuentra a la derecha de los argumentos. Este cuadro tiene el número

$$a_0(\bar{w}) = 2(x_1+\cdots+x_n+2n).$$

Si $n = 0$, entonces $a_0(\bar{w}) = 0$ y $b_0(\bar{w}) = 1$.

Ahora bien, tenemos

(i) $\qquad K(t,\bar{w},0) = \sigma_3(a_0(\bar{w}),b_0(\bar{w}),0).$

Además, como acabamos de ver, $K(t,\bar{w},z') = K(t,\bar{w},z)$ si $EtK(t,\bar{w},z)$, y $K(t,\bar{w},z') = F(t,K(t,\bar{w},z))$, en cualquier otro caso. Si introducimos una función recursiva primitiva φ definida como

$$\varphi(t,y) = \begin{cases} y\ , & \text{si } \ Ety \\ F(t,y)\ , & \text{en cualquier otro caso,} \end{cases}$$

entonces tenemos que

(ii) $\qquad K(t,\bar{w},z') = \varphi(t,K(t,\bar{w},z)).$

Las ecuaciones (i), (ii) muestran que la función K es recursiva primitiva.

3. El Predicado T_n, el cual es de grado $n+2$ $(n\geq 0)$, está definido por

(iii) $\qquad T_n t\bar{w}y \iff K(t,\bar{w},\sigma_{21}(y)) = K(t,\bar{w},(\sigma_{21}(y))') \wedge$
$$\sigma_{22}(y) = K(t,\bar{w},\sigma_{21}(y)).$$

Esta definición muestra que T_n es recursivo primitivo. En seguida demostramos tres resultados importantes.

(3.1) Si existe un z tal que $K(t,\bar{w},z) = K(t,\bar{w},z')$, entonces existe un y tal que $T_n t\bar{w}y$.

Demostración: Si $K(t,\bar{w},z) = K(t,\bar{w},z')$, sea $y = \sigma_2(z,K(t,\bar{w},z))$. Entonces $z = \sigma_{21}(y)$ y $K(t,\bar{w},\sigma_{21}(y)) = K(t,\bar{w},(\sigma_{21}(y))')$. Además $K(t,\bar{w},\sigma_{21}(y)) = K(t,\bar{w},z) = \sigma_{22}(y)$.

(3.2) Si existe un y tal que $T_n t\bar{w}y$, entonces existe un z tal que $K(t,\bar{w},z) = K(t,\bar{w},z')$, a saber $z = \sigma_{21}(y)$.

(3.3) Si M se detiene después de z_0 pasos y si $T_n t\bar{w}y$, entonces $\sigma_{22}(y) = K(t,\bar{w},z_0)$.

Demostración: Puesto que $T_n t\bar{w}y$ tenemos que $K(t,\bar{w},\sigma_{21}(y)) = K(t,\bar{w},(\sigma_{21}(y))')$. Además, por la propiedad (2.2) de K, tenemos que z_0 es el menor número z para el cual $K(t,\bar{w},z) = K(t,\bar{w},z')$. Por lo tanto, $z_0 \leq \sigma_{21}(y)$. Además, por la propiedad (2.1) de K, tenemos que $K(t,\bar{w},z_0) = K(t,\bar{w},z_0') = \cdots = K(t,\bar{w},\sigma_{21}(y))$, y $K(t,\bar{w},\sigma_{21}(y)) = \sigma_{22}(y)$ porque $T_n t\bar{w}y$. Por lo que $K(t,\bar{w},z_0) = \sigma_{22}(y)$.

4. Prueba del Teorema de Kleene.

Supongamos que la función n-aria f es calculada por la máquina M cuyo número de Godel es t. A continuación, representaremos a f con la ayuda de T_n. Sea z_0 el número de pasos después de los cuales M, aplicada tras $\bar{w}$, deja de operar. Según (3.1) existe un y tal que $T_n t\bar{w}y$. Además, $\mu y(T_n t\bar{w}y)$ es una aplicación del operador μ a un predicado regular. Naturalmente $T_n t\bar{w}(\mu y(T_n t\bar{w}y))$. Ahora bien, según (3.3), se sigue que $\sigma_{22}(\mu y(T_n t\bar{w}y)) = K(t,\bar{w},z_0)$ = número de Godel de la configuración terminal de M. De acuerdo con esto obtenemos, según (1), el valor de la función

$$f(\bar{w}) = W(K(t,\bar{w},z_0)) = W(\sigma_{22}(\mu y(T_n t\bar{w}y))).$$

Para terminar, introducimos una función recursiva primitiva U definida como

$$U(x) = W(\sigma_{22}(x)),$$

de esta manera obtenemos la representación

$$f(\bar{w}) = U(\mu y(T_n t\bar{w}y)).$$

Esto demuestra el Teorema de Kleene. En consecuencia:

TODA FUNCION TURING-CALCULABLE ES RECURSIVA.

(3.3) Si M se detiene después de z_0 pasos y si $T_n t\bar{w}y$, entonces $\sigma_{22}(y) = K(t,\bar{w},z_0)$.

Demostración: Puesto que $T_n t\bar{w}y$ tenemos que $K(t,\bar{w},\sigma_{21}(y)) = K(t,\bar{w},(\sigma_{21}(y))')$. Además, por la propiedad (2.2) de K, tenemos que z_0 es el menor número z para el cual $K(t,\bar{w},z) = K(t,\bar{w},z')$. Por lo tanto, $z_0 \leq \sigma_{21}(y)$. Además, por la propiedad (2.1) de K, tenemos que $K(t,\bar{w},z_0) = K(t,\bar{w},z_0') = \cdots = K(t,\bar{w},\sigma_{21}(y))$, y $K(t,\bar{w},\sigma_{21}(y)) = \sigma_{22}(y)$ porque $T_n t\bar{w}y$. Por lo que $K(t,\bar{w},z_0) = \sigma_{22}(y)$.

4. Prueba del Teorema de Kleene.

Supongamos que la función n-aria f es calculada por la máquina M cuyo número de Godel es t. A continuación, representaremos a f con la ayuda de T_n. Sea z_0 el número de pasos después de los cuales M, aplicada tras $\bar{w}$, deja de operar. Según (3.1) existe un y tal que $T_n t\bar{w}y$. Además, $\mu y(T_n t\bar{w}y)$ es una aplicación del operador μ a un predicado regular. Naturalmente $T_n t\bar{w}(\mu y(T_n t\bar{w}y))$. Ahora bien, según (3.3), se sigue que $\sigma_{22}(\mu y(T_n t\bar{w}y)) = K(t,\bar{w},z_0)$ = número de Godel de la configuración terminal de M. De acuerdo con esto obtenemos, segun (1), el valor de la función

$$f(\bar{w}) = W(K(t,\bar{w},z_0)) = W(\sigma_{22}(\mu y(T_n t\bar{w}y))).$$

Para terminar, introducimos una función recursiva primitiva U definida como

$$U(x) = W(\sigma_{22}(x)),$$

de esta manera obtenemos la representación

$$f(\bar{w}) = U(\mu y(T_n t\bar{w}y)).$$

Esto demuestra el Teorema de Kleene. En consecuencia:

TODA FUNCION TURING-CALCULABLE ES RECURSIVA.

A P E N D I C E   A

DEDUCCIONES

Un Procedimiento General se puede entender como un sistema de reglas a partir del cual se pueden producir *Deducciones*.

Definición (Deducción).

Una Deducción es una sucesión finita de palabras, sobre un alfabeto dado, que se ha producido mediante la aplicación de un sistema de reglas dadas de antemano. El número de palabras es llamado la longitud de la deducción.

En un sistema de reglas no está necesariamente especificado en qué orden deberán aplicarse las reglas. Además, no siempre es posible aplicar algunas de ellas en un momento dado. Existen sistemas de reglas, llamados superimpuestos, en los que hay más de un conjunto de reglas.[3] Dichos conjuntos están ordenados linealmente y para que una regla sea aplicada a una o varias palabras, éstas deberán haberse derivado mediante los conjuntos previos. Una palabra es *deducible* en el sistema si puede obtenerse mediante las reglas del último conjunto.

Aquí, podemos mencionar, una vez más, la conexión entre Procedimientos Generales y Algoritmos:

(i) Cada regla de un sistema describe un Algoritmo (terminante o no).

(ii) Cada deducción da origen a un Algoritmo que especifica el modo en que la deducción ha de producirse. Tal Algoritmo consiste en las reglas del sistema junto con una regla adicional que indica el orden en que aquellas deberán ser aplicadas.

Veamos el Diagrama siguiente

$$\left\{ \begin{array}{c} \text{Sistema de} \\ \text{Reglas} \end{array} \right\} \longrightarrow \text{Deducción} \longrightarrow \text{Algoritmo}$$

De hecho, un sistema de reglas debe considerarse como una fuente inagotable de Algoritmos.

Ejemplo (De un Sistema de Reglas):

El mínimo subconjunto K de números reales[1] que contiene a los números 37 y 15 y que es cerrado bajo la adición y cerrado bajo la multiplicación por $\sqrt{2}$, se puede obtener por medio del siguiente conjunto de reglas:

1) $37 \in K$.

2) $15 \in K$.

3) si $x, y \in K$, entonces $(x + y) \in K$.

4) si $x \in K$, entonces $(x \sqrt{2}) \in K$.

Un ejemplo de una deducción producida al aplicar este conjunto de reglas es:

(i) 15	(Por (2)).
(ii) 37	(Por (1)).
(iii) 52	(Por (3), usando (i) y (ii)).
(iv) $(52\sqrt{2})$	(Por (4), usando (iii)).
(v) $37 + (52\sqrt{2})$	(Por (3), usando (ii) y (iv)).

La cual es una deducción de longitud 5.

Ejemplo (De un Sistema de Reglas Superimpuesto):

El siguiente sistema está formado por tres conjuntos de reglas. El primer conjunto es usado para producir las variables

(1) Es pertinente aclarar que no deberíamos hablar de números, sino más bien de Representaciones Numéricas (NUMERALES). Es decir, objetos específicos que representan a los números escritos en notación decimal.

proposicionales, éstas son palabras sobre el alfabeto $A_1 = \{0, \not\}$ generadas por las siguientes reglas:

R_{11} : escríbase 0.

R_{12} : adhiérase $\not$ por la derecha.

De esta forma, algunas variables proposicionales son, por ejemplo: 0, 0/, 0//, 0///.

El segundo conjunto sirve para producir las fórmulas del cálculo proposicional, éstas son palabras especiales sobre el alfabeto $A_2 = \{ 0, /, \supset, (,) \}$ generadas por las siguientes reglas:

R_{21} : escríbase una palabra mediante el sistema $\{R_{11}, R_{12}\}$.

R_{22} : si las palabras ρ_1 y ρ_2 han sido generadas, escríbase $(\rho_1 \supset \rho_2)$.

De esta forma, algunas fórmulas del cálculo proposicional son, por ejemplo:

0/, 0//, (0// ⊃ 0/), ((0// ⊃ 0/) ⊃ 0//)).

El tercer conjunto sirve para obtener Tautologías. Las Tautologías son fórmulas especiales generadas por las siguientes reglas:

Si ρ_1 , ρ_2 , ρ_3 han sido obtenidas mediante el sistema $\{ R_{11} , R_{12} , R_{21} , R_{22} \}$ entonces

R_{31} : escríbase $(\rho_1 \supset (\rho_2 \supset \rho_1))$.

R_{32} : escríbase $((\rho_1 \supset (\rho_1 \supset \rho_2)) \supset (\rho_1 \supset \rho_2))$.

R_{33} : escríbase $((\rho_1 \supset \rho_2) \supset ((\rho_2 \supset \rho_3) \supset (\rho_1 \supset \rho_3)))$.

R_{34} : si $(\rho_1 \supset \rho_2)$ y ρ_1 han sido generadas, en este último conjunto, entonces escríbase ρ_2.

APENDICE B

EJEMPLOS DE TURING-CALCULABILIDAD Y DE TURING-DECIDIBILIDAD

En las definiciones de Turing-Calculabilidad de una Función y de Turing-Decidibilidad de un Predicado se requiere que la máquina que ejecuta la tarea pueda ser aplicada sobre un cuadro arbitrario de la cinta. Como veremos, en ambos casos, dicha máquina se puede definir con la ayuda de las máquinas L_d, S (definidas en la sección IV.6) y de otra máquina M' que es aplicada sobre un cuadro particular (para nuestros fines, este cuadro es el primer cuadro vacío a la derecha de los argumentos).

Partimos de la suposición de que conocemos una máquina M' que realiza el cálculo del valor de una función f n-aria (n≥1) al ser aplicada sobre el primer cuadro vacío a la derecha de los argumentos, entonces podemos describir una máquina M, con la ayuda de M', que calcula a f en el sentido de la definición de Tuting-Calculabilidad.

Teorema 1. Sea f una función n-aria (n≥1) definida para todas las palabras sobre un alfabeto $A = \{a_1, \ldots, a_r\}$ y cuyos valores son palabras sobre este alfabeto[1]. Sea M' una máquina sobre A tal que si escribimos sobre la cinta vacía los n argumentos $\rho_1, \ldots, \rho_n$ y aplicamos M' sobre el primer cuadro vacío a la derecha de éstos, entonces M' deja de operar (después de un número finito de pasos) tras haber calculado el valor $f(\rho_1, \ldots, \rho_n)$ de la función. Entonces f es Turing-Calculable y una Máquina de Turing M que calcula a f esta dada por

$$M = SL_dM' .$$

(1) se utiliza la letra r (en lugar de la n) para representar el número de letras del alfabeto, debido a que la n (en este caso) representa el número de argumentos de la función.

La afirmación del Teorema 1 es muy clara. Puesto que S busca un cuadro que este marcado por los argumentos; a continuación L_d localiza el primer cuadro vacío a la derecha de éstos; y para terminar M' es aplicada para calcular $f(\rho_1,\ldots,\rho_n)$.

De la misma manera podemos probar el siguiente Teorema:

Teorema 2. Sea R una relación n-aria ($n\geq 1$) en el dominio de palabras sobre el alfabeto $A = \{a_1,\ldots,a_r\}$. Sea M' una máquina tal que si escribimos sobre la cinta vacía una n-ada $(\rho_1,\ldots,\rho_n)$ de palabras sobre A y aplicamos M' sobre el primer cuadro vacío a la derecha de éstas, entonces M' deja de operar (después de un número finito de pasos) sobre el símbolo a_i o a_j (con $a_i, a_j \in A \cup \{a_0\}$ e $i\neq j$) ya sea que $(\rho_1,\ldots,\rho_n)$ estén en la relación R o no. Entonces R es Turing-Decidible y una Máquina de Turing que decide a R está dada por $M = SL_dM'$.

Ejemplos de Funciones Turing-Calculables.

Vamos a considerar sólo funciones cuyos argumentos y valores son Números Naturales. Las máquinas de Turing siguientes son máquinas sobre el alfabeto $\{ / \}$, de modo que el número natural n será representado por $(n+1)$ trazos (según se describió en la sección I.3). Además, supondremos que estas máquinas son aplicadas sobre el primer cuadro vacío a la derecha de los argumentos dados. Este cuadro es el cuadro inicial de la máquina M' mencionada en el Teorema 1 de esta sección, por lo que sólo se necesita aplicar dicho Teorema para encontrar las máquinas que calculan a las funciones consideradas en el sentido de la definición de Turing-Calculabilidad.

(1) La función sucesor $S(x)$ es calculada por $/D$.

(2) La función suma $x+y$ (denotada S_m) es calculada por

$$S_m = B_i/B_dIa_0Ia_0 .$$

(3) La función $f(x)=2x$ es calculada por CS_m .

Ejemplo de Relación Turing-Decidible.

La propiedad de un número natural x $(x \geq 0)$ de ser divisible por un número natural fijo n $(n \geq 1)$ es Turing-Decidible. Vamos a considerar una máquina M' sobre el alfabeto $\{a_1\}=\{/\}$, de modo que el número natural m será representado por $(m+1)$ trazos. Además, dicha máquina lleva a cabo el proceso de decisión (con $a_i=*$ y $a_j=/$) al ser aplicada sobre el primer cuadro a la derecha del argumento; y deja de operar después de un número finito de pasos sobre $/$ si n no divide a x o sobre $*$ si n divide a x. De modo que aplicando el Teorema 2 de esta sección se obtiene una máquina que decide a esta propiedad en el sentido de la definición de Turing-Decidibilidad.

La máquina M' esta definida por el siguiente diagrama:

$$
\begin{tikzcd}
II^{n} \arrow[loop right, "1"] \arrow[d, "0"] \\
D^{n-1}M^{0}
\end{tikzcd}
$$

A P E N D I C E C

MANUAL PARA LA UTILIZACION DEL PROGRAMA

A continuación se mencionan las características principales del programa, lo cual servirá para comprender su funcionamiento y poder utilizarlo debidamente.

El programa simula el trabajo de las Maquinas de Turing. Esto es, al programa se le dan como datos los elementos que definen una matriz que representa una cierta Maquina de Turing y este va realizando el trabajo de la Máquina paso por paso hasta alcanzar una configuracion terminal[1].

El Menu Principal del Programa es:

1.- Dar de alta una tabla y activar la maquina.

2.- Leer una tabla del disco y activar la maquina.

3.- Visualizar una tabla en pantalla.

4.- Pasar al Menu de Digráficas.

Ahora bien, al dar de alta una tabla se debe recordar que los actos posibles de una Máquina de Turing son:

/ : La impresión del simbolo "/" en la ventana.

***** : Asignarle a la ventana el simbolo vacio.

D : Mover la ventana un cuadro hacia la derecha.

I : Mover la ventana un cuadro hacia la izquierda.

P : Detenerse (Parar).

(1) en el programa, una máquina se detendrá:

(a) si durante el proceso se encuentra un acto de parada.

(b) si estando en el primer cuadro de la semicinta (que se ve en la pantalla) se encuentra la instrucción de moverse a la izquierda.

(c) si estando en el último cuadro de la semicinta (que se ve en la pantalla) se encuentra la instrucción de moverse a la derecha.

Los casos (b) y (c) son consecuencia de que en la pantalla estamos limitados a usar UNA SEMICINTA FINITA.

Además, el primer dato que debemos darle al Programa es el número E de Estados que tiene la maquina que vamos a dar de alta. Posteriormente damos como datos los elementos de la tercera y cuarta columna de la matriz que define a la maquina. Esto es, el programa nos pide para cada estado i $(0 \leq i \leq E-1)$ los siguientes datos (uno por uno):

ACTO[i,*] =

ESTADO[i,*] =

ACTO[i,/] =

ESTADO[i,/] = .

donde ACTO[i,j] representa el Acto a realizar por la maquina si ésta se encuentra en el estado i y en la ventana esta el símbolo j (j = * o /); y ESTADO[i,j] representa el Nuevo Estado al que pasa la maquina si esta se encuentra en el estado i estando en la ventana el simbolo j (j = * o /).

Despues de haber dado de alta una Máquina, el Programa nos permite grabar los elementos que la definen preguntandonos:

SE GRABA EN DISCO ? (S/N).

Nota: Es importante decir que si se elige grabar en disco una maquina, el Programa nos preguntara: COMO SE LLAMA LA MAQUINA ?. De modo que en ese momento se le debe asignar un nombre a dicha Máquina.

De este modo el Programa nos permite:

(i) leer los datos de una matriz (tabla) ya grabada,e ir ejecutando el trabajo de ésta, o

(ii) visualizar en la pantalla una tabla ya grabada.

El programa maneja sólo máquinas que trabajan sobre el alfabeto $A = \{a_1\} = \{/\}$. Es decir, las maquinas utilizadas en dicho programa solo utilizan los simbolos a_0 = * y a_1 = /. Ademas, vamos a considerar sólo máquinas cuyos estados estén numerados por 0,1,...,etc. Siendo 0 el estado inicial, lo cual (como vimos) no constituye una limitación.

Dicho lo anterior, para definir una digrafica en el programa se deben realizar los siguientes pasos:

1.- Numerar las maquinas que intervienen en la digrafica considerando que: Si una maquina aparece mas de una vez, entonces las repeticiones deberan contarse como si fueran máquinas distintas. Ademas, las maquinas se van a numerar 1,2,3,...,etc. y no $M_1, M_2, M_3, \ldots$,etc.

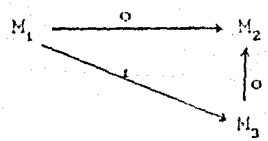
2.- Al describir una digráfica en la cual intervengan r maquinas (incluyendo las repeticiones) debemos tomar en cuenta lo siguiente: si de un vertice k (con $1 \le k \le r$) no sale ninguna arista con el numero j (con j=0 o j=1 o ambos) entonces definiremos dichas aristas de modo que conecten al vertice k con un vertice auxiliar que llamaremos CERO. De modo que la digrafica en el programa quede definida de manera que de cada vertice involucrado en la digrafica original salen dos aristas (a saber, la arista 0 y la arista 1 correspondientes a los dos posibles simbolos 0 y 1 respectivamente). Si, por otro lado, de cada vertice de la digráfica original salen dos aristas (correspondientes a los símbolos 0 y 1) entonces la digrafica definida en el programa no sufre ninguna alteracion.

   Este proceso de completar se realiza porque al dar de alta una digrafica (en el programa) se debe indicar *forzosamente* a dónde estan conectadas la arista 0 y la arista 1 de cada maquina involucrada en tal digrafica.

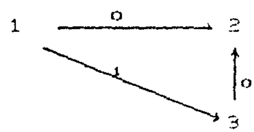
Nótese que al completar la digráfica, conectando al vértice CERO (como se ha indicado), no se produce ninguna alteracion en la digráfica original puesto que: la maquina que representa a la digrafica obtenida despues de la complementación es *intercambiable* con la maquina que representa a la digrafica original, considerando que el vertice CERO utilizado al completar la digrafica para el programa representa a M^0 al completar la digráfica original manteniendo la notación $M_1, M_2, \ldots$,etc.

Así, por ejemplo:

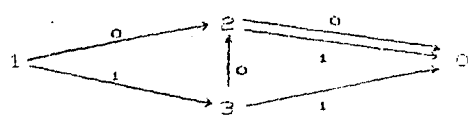
La digráfica original



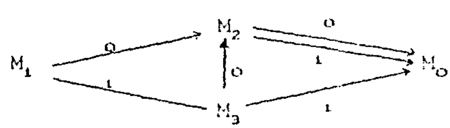
después del primer paso queda:



y después del segundo paso queda definida, en el programa, como:



Ahora bien, esta digráfica definida en el programa también representa a la digráfica:



la cual es intercambiable con la digráfica original.

Comentario: Dada una digráfica, nótese que si el único vértice del cual no sale ninguna arista es el que representa a M^0, entonces la digráfica definida para el programa no sufre ninguna alteración.

Ahora bien, como los datos que definen a una digrafica, es decir

i) el número de vértices que intervienen en ella y

ii) los vértices a los que estan conectadas las aristas 0 y 1 de cada máquina involucrada en la digráfica,

pueden ser grabados en el disco, el programa permite leer una digrafica ya grabada y activarla.

Nota: Las tablas de las Máquinas involucradas en una digráfica deben estar grabadas en el disco que contiene al Programa.

Ligador de tablas de una digráfica.

El programa también permite "ligar" las tablas de las máquinas de una digráfica para formar una sola tabla. Es decir, ya teniendo los datos que definen una digráfica, la opción: *Ligar las tablas de una digráfica* [1] construye la tabla que representa a la máquina de Turing que realiza el trabajo de la digráfica (tal y como se explicó en la sección IV.5, cuando se definió la máquina de Turing M representada por una digráfica).

Comentario: *Nótese que el LIGADOR es un constructor de Máquinas de Turing que Calculan Funciones Recursivas. Puesto que, Toda Función Recursiva es Turing-Calculable y el LIGADOR nos permite construir la tabla que representa a la máquina que calcula a dicha función.*

(1) A esta opción también la llamaremos LIGADOR.

Nota: Al grabar una digrafica en el disco se le debe asignar un nombre con la extension ".DIG". Esto es porque al grabar una tabla en el disco, a ésta tambien se le asigna un nombre; y la extension mencionada sirve para diferenciar los nombres de las tablas y los nombres de las digraficas. Al "ligar" las tablas de una digráfica para obtener una sola tabla, esta tambien podra ser grabada y se le debe asignar un nombre (sin extension).

Según lo que hemos visto, al pasar al menu de digraficas, tendremos:

MENU DE DIGRAFICAS

1.- Dar de alta una digrafica y activarla.
2.- Leer una digrafica del disco y activarla.
3.- Ligar las tablas de una digrafica.
4.- Regresar al menu principal.

A continuacion se explica como llevar a cabo la ejecucion del Programa: El Programa esta escrito en turbo pascal 3 y se llama TURING. Ahora bien, lo primero que debemos hacer para ponerlo en marcha es: (una vez colocado el disket, que contiene al Programa[1], en la computadora) teclear la palabra TURBO y presionar <<ENTER>>. En seguida aparecera el menu de Turbo Pascal 3, del cual elegimos la opcion R (Run). Ahora debemos teclear el nombre del Programa y presionar <<ENTER>>. En este momento el Programa es compilado e inmediatamente despues observamos en la pantalla la presentacion del mismo, que es un letrero a colores que dice "TURING". Este letrero sirve para definir la semicinta, el trazo (símbolo del alfabeto), el blanco (símbolo vacío) y la ventana que se utilizaran al simular el trabajo de una Máquina de Turing. A continuacion, para pasar al menú principal, presionamos <<ENTER>>.

(1) El Programa en su totalidad se encuentra grabado en un solo disco, el cual se proporciona adicional al presente trabajo.

Al escribir los argumentos necesarios para activar una máquina se debe tomar en cuenta:

(i) que las máquinas definidas en el Programa sólo actúan sobre números naturales, y

(ii) que el número natural n está representado por $n+1$ trazos.

Para escribir los argumentos se usan las teclas:

/ para representar un trazo[1],

* para representar al símbolo vacío[2],

→ para mover la ventana[3] un cuadro a la derecha,

← para mover la ventana un cuadro a la izquierda,

↑ para mover la ventana un cuadro hacia arriba, y

↓ para mover la ventana un cuadro hacia abajo.

Una vez escritos los argumentos se debe presionar <<ENTER>> para activar la máquina.

(1) Recordar que se tiene la correspondencia: / ⟶ 0, // ⟶ 1, /// ⟶ 2, //// ⟶ 3, ...etc.

(2) En el Programa, un cuadro se considera vacío si no tiene ningún símbolo escrito en él. En consecuencia, al momento de escribir los argumentos necesarios para activar una máquina se considera que todos los cuadros de la semicinta estan vacíos.

(3) Al cuadro en inspección lo llamamos VENTANA.

Ahora bien, podemos ir observando el trabajo de la máquina de las siguientes formas:

(i) lentamente (presionando la tecla del <<espacio>> para ejecutar cada paso) o

(ii) en forma rápida (presionando la tecla F1).

Es importante mencionar que las teclas F1 y <<espacio>> funcionan como interruptores, de modo que se puede alternar la velocidad de ejecución de la máquina.

Una vez activada una máquina, si se presiona la tecla <<ENTER>> la máquina dejará de operar.

Al dejar de operar la máquina, presionando <<ENTER>> se tendrá la opción de volver a activarla o de pasar al menú principal.

Considero que con lo explicado en este apendice, y con las indicaciones que aparecen en el Programa, no habrá problema para su utilización.

B I B L I O G R A F I A

1.- Borlan Internacional Inc. Turbo Pascal version 3.0. Reference Manual. California 1985.

2.- CONACYT. Revista Ciencia y Desarrollo número 64. Artículo *"La Recursividad del Universo"* por Enrique Calderón y José Negrete. México, D.F., 1985.

3.- Hermes, Hans. Enumerability, Decidability, Computability. Springer-Verlag, New York, 1969.

4.- Kleene, Stephen C. Mathematical Logic. Wiley 1967.

5.- Korfhage, Robert R. Lógica y Algoritmos. Editorial Limusa. México, D.F., 1985.

6.- Torres Alcaraz, Carlos. Los Teoremas de Godel. Tesis para obtener el título de Maestría en Matemáticas. Facultad de Ciencias, UNAM, 1989.

7.- Torres Alcaraz, Carlos. Notas de clase para la materia de Lógica Matemática III. Departamento de Matemáticas. Facultad de Ciencias, UNAM, 1986.