UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

LA LOGICA MATEMATICA APLICADA A LOS ALGORITMOS Y TECNICAS DEL CALCULO PROGRAMADO

T E S I S

QUE PARA OBTENER EL TITULO DE:

M A T E M A T I C O

P R E S E N T A :

ANTONIO LABASTIDA MORALES

MEXICO, D. F.

1982

CIUDAD UNIVERSITARIA

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS
División de Estudios Profesionales
Exp. Núm. 55

Of. Núm. B- 1527

ING. FERNANDO RUEDA PUERTO
Jefe del Departamento de
Expedición de Documentos
Coordinación de la Administración Escolar
Universidad Nacional Autónoma de México
P r e s e n t e

En virtud de que el señor Antonio Labastida Morales, pasante de la Carrera de MATEMATICO ha cumplido todos los requisitos para presentar su examen profesional, a continuación comunico a usted el Jurado respectivo:

MAT. JOSE ALFREDO AMOR MONTAÑO	Presidente
MAT. LOURDES GUERRERO ZARCO	Vocal
MAT. LEOBARDO PLATA PEREZ	Secretario
MAT. RAFAEL ROJAS BARBACHANO	Suplente
MAT. MARIA GUADALUPE ELENA IBARGUENGOITIA GONZALEZ	Suplente

El trabajo aprobado como tesis profesional es:

"LA LOGICA MATEMATICA APLICADA A LOS ALGORITMOS Y TECNICAS DEL CALCULO PROGRAMADO".

Reitero a usted las seguridades de mi consideración más distinguida.

Atentamente,
México, D.F., 6 de septiembre de 1982
LA SECRETARIA DE LA
DIVISION DE ESTUDIOS PROFESIONALES

DRA. SYLVIA DE NEYMET URBINA

SNU'avg.

I N D I C E

LA LOGICA MATEMATICA APLICADA A LOS ALGORITMOS Y TECNICAS DEL CALCULO PROGRAMADO

P R O L O G O

Al tratar el tema de la Lógica, tema tan vasto como su desarrollo mismo al través de los siglos, uno puede agrupar dicho tema, según el contexto que se dé, en tres categorías fundamentales:

1o. La Lógica tratada por filósofos, tendente a enfatizar el uso y significado de la misma en la conversación cotidiana.

2o. La Lógica Matemática, que prescinde de las interpretaciones del grupo anterior y se concentra en un enfoque formal algebraico.

3o. El tema denominado Algebra Booleana, tratada en su aplicación a la Ingeniería en la teoría de los circuitos.

En el presente trabajo, he tratado de encerrar el tema de la Lógica dentro del contexto de su aplicación a la ciencia de la comunicación, acorde a los modernos dispositivos físicos existentes en la actualidad para el tratamiento automático de la información; me refiero a las computadoras. Así, dado que el tema que trato es el de la comunicación —entre personas o entre personas y computadoras— el presente trabajo sienta su base en la muy importante idea de ALGORITMO.

Los temas desarrollados entonces toman su substancia del 2o. y 3er. grupo esencialmente de nuestra clasificación anterior. Abarco el aspecto histórico del primer grupo en el Capítulo I con una breve reseña del desarrollo de la lógica hasta nuestros días, comparando su desarrollo con el de la invención en occidente de "máquinas de calcular" hasta las computadoras actuales.

CAPITULO I

INTRODUCCION HISTORICA

La evolución de la Lógica comienza en la Grecia antigua, principalmente con el filósofo Aristóteles (384-322 A.C.) y otros griegos de la antigüedad.

Como en la Geometría de Euclides y otros campos, las ideas griegas sobre la lógica dominaron por completo el campo durante 2,000 años. Luego, alrededor de 1666, Leibniz (1646-1716) propuso en varios trabajos lo que llamó el "cálculo racional" o "Logística". Aunque Leibniz nunca desarrolló estas ideas con gran extensión, sus escritos contienen el germen de gran parte de lo que hoy conocemos como Lógica Matemática.

Como dato curioso, vale notar que, análogo a la Geometría de Euclides y la Lógica de Aristóteles, el Cálculo Diferencial avanza simultáneamente y luego, como la lógica, duerme largamente hasta el Siglo XIX.

En 1642 Blaise Pascal (1623-1662) construye la primera máquina de sumar. En 1671 Leibniz construye una máquina que podía también multiplicar. Incidentalmente, Leibniz introdujo también en la terminología matemática la palabra "función", y fue el primero en occidente en reconocer la importancia del sistema numérico binario.

El siguiente acontecimiento importante en nuestra historia seda en 1812, cuando el inventor inglés Charles Babbage (1792-1871) diseña una "máquina de diferencias" cuyo propósito era ayudar a la construcción de tablas matemáticas. El gobierno

británico subvencionó el proyecto; pero luego de varios años - de inútiles esfuerzos en la construcción del aparato, lo abandonó. En 1830 Babbage nuevamente propuso una "máquina analítica" que tenía un almacenaje mecánico para los datos e instrucciones análogas al almacenaje electrónico de las computadoras- de hoy en día. La falla en esta máquina, como la de la "máquina de diferencias", se debió no a errores en su concepción, sino a que las herramientas y la tecnología de entonces no eran- adecuadas para proporcionar los componentes de precisión requeridos.

Aunque el campo de la computación yace inmóvil durante la si-- guiente centuria, el trabajo en Lógica avanza rápidamente. En 1847, Augusto De Morgan (1806-1871) publica un tratado: FORMAL LOGIC. Al año siguiente, George Boole (1815-1864) escribió -- THE MATHEMATICAL ANALYSIS OF LOGIC, obra a la que siguió, en - 1854, AN INVESTIGATION OF THE LAWS OF THOUGHT. Aunque ninguno de estos trabajos contenía todos los elementos básicos de la - lógica tal como la entendemos hoy (Boole, por ejemplo, considera- raba solamente el uso de "o" como disyunción exclusiva al desarrollar su sistema algebraico), en los siguientes 50 años un - gran número de excelentes lógicos expandió esta base hasta que Gottlob Frege la presenta esencialmente en su forma actual. Este período en el desarrollo de la Lógica culminó en 1910-1913- con la publicación de los PRINCIPIA MATHEMATICA de Alfred North Whitehead (1861-1947) y Bertrand Russell (1872-1970). Fue este un intento monumental de completar un programa que Leibniz había sugerido: el de proporcionar una base lógica para todas - las matemáticas. El trabajo incluye Teoría de Conjuntos en la forma en que la habían desarrollado George Cantor (1845-1918)- y otros, y algunas partes de la Geometría; pero nunca se llevó más lejos.

Por el tiempo de Whitehead y Russell, la Lógica Matemática ha-

bía alcanzado una saludable adolescencia y crecía rápidamente en madurez. Desde ese tiempo el estudio de la Lógica se ha desarrollado en diferentes direcciones, siendo la que trataré en este trabajo la que nos lleva al concepto de algoritmo y a las computadoras. Pero aún hay más; durante la última mitad del siglo XIX los matemáticos habían estado explotando y extendiendo el "método axiomático" —el de enunciar axiomas y luego derivar teoremas formalmente de ellos —.

Aunque gran parte de este trabajo se había hecho en la Teoría de Conjuntos, la Geometría y la Lógica, a fines de siglo parecía completamente razonable a muchos matemáticos que, con el tiempo, todas las distintas partes de las matemáticas podrían axiomatizarse. La posición extremista en esta cuestión era que no sólo las distintas partes podrían axiomatizarse, sino que las matemáticas en su conjunto podrían realmente axiomatizarse de un modo consistente y completo relativamente sencillo. El matemático alemán David Hilbert (1862-1943) tenía precisamente esos puntos de vista. En 1900, Hilbert propuso una lista de 23 problemas cuya solución consideraba esencial para el avance de las matemáticas. Muchos de estos problemas ya se han resuelto, muchos han dado lugar a otros problemas profundos; algunos siguen aún sin solución. Se ha demostrado que unos pocos de ellos dependen fuertemente de conceptos lógicos y por ello, han influido en el desarrollo de la lógica.

El programa que Hilbert tenía en mente era el desarrollo de un sistema Lógico-matemático dentro del cual estuvieran inmersas todas las matemáticas y que fuera demostrable su consistencia, en el sentido de que sería posible probar que de los axiomas de este sistema nunca se podría llegar a probar una contradicción como teorema. Pero en 1931, para decepción de los matemáticos como Hilbert, este programa recibió un golpe mortal. Un joven austriaco, Kurt Gödel (1906-1978) publicó un artículo —

conteniendo el siguiente resultado:

> Cualquier sistema formal que sea suficientemente general para contener completamente la teoría elemental de los números, debe también contener fórmulas que no se pueden ni probar ni refutar dentro del sistema.

Lo que Gödel mostró fue que no importa cuán fuerte sea un sistema formal que se invente, de ser consistente siempre habrá problemas que podrán ser formulados dentro del sistema, pero que no podrán resolverse dentro de él, o sea, será incompleto. Esto quiere decir que, en nuestro contexto matemático, hay problemas de las matemáticas que, forzosamente, nunca podrán resolverse, no simplemente porque no se haya descubierto cómo resolverlos, sino porque el hombre parece que es inherentemente incapaz de encontrarles una solución.

Esto pone al matemático en una situación desesperante y algo divertida, "como el gato que persigue su cola". Tal parece que es propio de la condición humana el llegar a situaciones absurdas y perseverar en las mismas.

Como Post ha escrito: ".... como las pruebas clásicas de insolubilidad, estas pruebas son de insolubilidad por medio de instrumentos dados. Lo que es nuevo, es que.... estos instrumentos, en realidad, parece que son los únicos instrumentos de que el hombre dispone".

Pero no todo fue decepción y derrumbe, si bien el trabajo de Gödel destruyó el programa de Hilbert, condujo en cambio al rápido desarrollo de un campo descuidado de las matemáticas: el de determinar qué métodos son válidos en la resolución de problemas. Las cuestiones básicas aquí son exactamente lo que se quiere decir cuando afirmamos que tenemos un algoritmo para re

solver un problema, que podemos "calcular efectivamente" una - función o que podemos "enumerar efectivamente" un conjunto.

En los años treintas, un cierto número de excelentes matemáticos y lógicos se hicieron estas preguntas, con el resultado de que fueron varios los conceptos que se propusieron como res- - puestas satisfactorias. Entre los más importantes de estos - conceptos —sólo por mencionarlos— están los de las funciones recursivas — introducidas por Gödel —, λ - definibilidad — debido a Church — y computabilidad tal como lo postulan Turing e, independientemente, Post. Mientras que estos conceptos son superficialmente diferentes por completo, el hecho de que fueran equivalentes era cosa sospechada por quienes los habían -- creado o los manejaban, y demostrada efectivamente por primera vez en dos artículos de Kleene y Turing.

Después de esta explosión de actividad, el trabajo en el campo de la computabilidad se hizo gradualmente más ligado a los conceptos más finos de la teoría. En tanto, el mundo de las computadoras nacía lentamente. Mientras que las computadoras analógicas habían sido ya discutidas y construidas desde hacía varios años (las máquinas analógicas son sistemas físicos que se comportan de modo semejante a otros sistemas físicos abstractos; miden y representan los números con magnitudes físicas tales como presión, temperatura, voltaje y corriente. Por ejemplo, una regla de cálculo es un mecanismo analógico en el que la longitud sirve de análogo del logaritmo de los números) y - la tarjeta perforada había sido inventada por Herman Hollerith nada menos que en 1886, las computadoras digitales no aparecen sino hasta 1940 (las máquinas digitales son sistemas físicos - que cuentan utilizando series de dígitos para representar un - número. Por ejemplo, una calculadora mecánica de escritorio es una máquina digital con ruedas dentadas que sirven de dígitos). A partir de esta fecha el mundo de las computadoras crece ace-

leradamente, dando dirección e ímpetu a estudios en linguística, teoría de algoritmos y lógica. El desarrollo de la teoría de los autómatas ha sido motivada a menudo por un deseo de - - construir modelos matemáticos de computadoras digitales, y los trabajos recientes en este campo toman en cuenta cada vez más las características estructurales de estas máquinas.

En los capítulos siguientes prepararé el terreno para mostrar al final el tipo de trabajos hechos en Lingüística, en Teoría de algoritmos (Trabajo de Markov), la versión de Turing y el - uso de las computadoras en la construcción de algoritmos para la demostración de algunos teoremas de la Lógica Matemática.

CAPITULO II

NOCIONES PRELIMINARES

a) Conjuntos, relaciones y mapeos

b) Algebras Booleanas

INTRODUCCION AL CAPITULO II

El punto de partida de toda estructura matemática supone la necesidad de razonar en forma válida acerca de cosas trascendentes y singularmente importantes. Esto implica que ha de existir absoluta claridad y distinción en todo lo concerniente al razonamiento deductivo válido, esto es: significado de los términos usuales, uso de proposiciones, definiciones, teoremas, - eliminación de las complicaciones y contingencias relativas a- la vida cotidiana, reconocimiento y abandono de las falacias y ambigüedades, etc. Todo ello obliga a una gran simplificación y a la introducción de un simbolismo adecuado e inequívoco, -- que permita razonar válidamente mediante reglas fijadas con -- claridad, aunque sean arbitrarias.

Dos temas entrañablemente unidos que dan este paso son: la Lógica Simbólica (o Matemática) y la Teoría de Conjuntos.

En este capítulo desarrollaré los conceptos e ideas básicos de la Teoría de Conjuntos desde el concepto indefinible de conjunto hasta las álgebras Booleanas. La idea de conjunto forma un concepto unificante en la matemática moderna y es la base del- tema de este trabajo. Así pues, en la primera parte (a) se cubren los conceptos básicos de esta teoría que utilizaré des- - pués, en la segunda parte (b) se discuten las Algebras Booleanas. Estas constituyen una simple abstracción de las álgebras de conjuntos y son importantes por dos aspectos:

En su aplicación: Se utilizan en el diseño de las computado-- ras digitales; y

En su concepto: Proporcionan un nexo entre la Teoría de Conjuntos y la Lógica elemental.

a) CONJUNTOS, RELACIONES Y MAPEOS.

1. Introducción y notación.
2. Algunos conjuntos especiales.
3. Combinación de conjuntos.
4. Algebra de conjuntos.
5. Productos, relaciones y mapeos.
6. La relación subconjunto.

1.- INTRODUCCION Y NOTACION

Un punto en el que muchas discusiones formales o informales de cualquier tópico pueden terminar, es la cuestión de la definición de los términos básicos; y nuestro tema no es la excepción. Así pues, partiremos de ciertos "indefinibles" para, a-partir de ahí, desarrollar el tema.

Nuestros términos indefinidos serán las palabras: "conjunto" y "elemento" y la frase "es un elemento de".

Podemos pues discutir estas palabras y decir informalmente lo-que queremos que signifiquen, pero no las definiremos.

Por un conjunto A entendemos una colección de objetos, a los -que llamaremos los elementos de A, (Nótese que no nos ayuda definir un conjunto como "una colección de objetos", pues luego-deberíamos decir qué es lo que entendemos por la expresión "colección de objetos"). Estos objetos pueden ser cosas cualesquiera: conceptos, símbolos, conjuntos, etc.

Supondremos que siempre que escribimos: "x es un elemento de -A" podemos también escribir "x es un elemento" y "A es un conjunto". La única restricción que establecemos es que tenga -- sentido preguntar "¿es x un elemento de A?", y que para cada x y cada A la pregunta pueda de algún modo ser contestada por -- "si" o "no".

Es bien conocido que una teoría formal de conjuntos construida sobre este amplio marco lleva a situaciones paradójicas. Si -uno desea construir una teoría de conjuntos debe evitar que cosas tales como "la colección de todos los conjuntos" sea un -- conjunto. Como nuestro interés primario no va en esta direc-- ción, no nos meteremos en las complejidades a las que, cuando-

estas paradojas se quieren evitar, hay que enfrentarse; pero - supondremos que los conjuntos en nuestro contexto se comportan bien.

Respecto a la notación adoptaremos la usual.

- A los conjuntos los representaremos por letras mayúsculas A, B, C,.....
- A los elementos por letras cursivas minúsculas a, b, c,..
- La frase "es un elemento de" por $\in$
- Y la frase "no es un elemento de" por $\notin$

2.- ALGUNOS CONJUNTOS ESPECIALES

Como los elementos que pueden estar (o no estar) en un conjunto pueden ser de cualquier tipo concebible, resulta deseable - la introducción de un conjunto universal.

Por ejemplo, si consideramos $A = \{ x \mid 2x = 5 \text{ ó } x = 0 \}$, es claro que A debe ser un conjunto de números.

Pero el símbolo $\{ x \mid 2x = 5 \text{ ó } x = 0 \}$ no nos dice que x sea un número: en este plan detallista, es entonces concebible buscar x que satisfacieran $2x = 5$ entre cosas tales como triángulos,- bocas de riego y pinturas abstractas, sin atender a si esta búsqueda tiene algún sentido. Es desde luego permisible escribir $A = \{ x \mid x \text{ es un número y } 2x = 5 \text{ ó } x = 0 \}$, mas si se van a manejar muchos conjuntos de números, esto nos obliga a una repetición tediosa. Lo que hacemos entonces es enunciar que "el-- conjunto universal U es el conjunto de números", y ya, tácitamente, se supone que cualquier elemento al que nos refiramos - tiene la propiedad de que es un elemento del conjunto particu-

lar prescrito como conjunto universal.

Así pues, un conjunto universal no contiene todo, sino que es, más bien, el universo del discurso en un momento determinado.

Hay también otra razón para insistir en la especificación de un conjunto universal, a saber, la de que los elementos de un conjunto dependen del conjunto universal.

Por ejemplo, $A = \{ x \mid 2x = 5 \ \& \ x = 0 \}$

Si U es el conjunto de los números reales, resulta

$$A = \{ 0, 2.5 \}$$

pero si U es el conjunto de todos los enteros, entonces

$$A = \{ 0 \} .$$

Sin embargo, si U es el conjunto de todos los enteros impares, entonces A no tiene elemento alguno. Esto nos muestra la necesidad de otro conjunto especial $\emptyset$, llamado el conjunto vacío, que es el conjunto que no tiene elemento alguno. Puede caracterizarse como $\{ x \mid x \neq x \}$.

Es frecuente que estemos intersados en sólo ciertos elementos de un conjunto determinado, esto nos lleva a la siguiente definición.

Definición 2.1

> Decimos que A es un subconjunto de B, lo que escribimos $A \subseteq B$, si y sólo si todo elemento de A es también un elemento de B; i.e. si siempre que $x \in A$, entonces también $X \in B$.

Observamos, en particular, que un conjunto A es siempre un subconjunto de él mismo, ya que siempre que $X \in A$, entonces $X \in A$. - El conjunto vacío $\emptyset$ es también subconjunto de cualquier conjunto dado A, ya que la condición de la definición se satisface - por vacuidad.

Definición 2.2

Dos conjuntos A y B son iguales, $A = B$, si y sólo si - $A \subseteq B$ y $B \subseteq A$ (Es decir, si y sólo si todo elemento de A- es un elemento de B y viceversa), o bien, si y sólo si A- y B tienen los mismos elementos.

Ejemplo.- Si U es el conjunto de números reales y $A = \emptyset$ y $B = \{ x \mid x^2 + 1 = 0 \}$ entonces $A = B$

Definición 2.3

A es un subconjunto propio de B, $A \subsetneq B$, si $A \subseteq B$ y $B \neq A$.

Ejemplo.- Si U es el conjunto de los números complejos y $A = \{i\}$ y $B = \{ x \mid x^2 + 1 = 0 \}$ entonces $A \subsetneq B$.

Definición 2.4

Sea U el conjunto universal y A un conjunto cualquiera. - El complemento de A, A^c, está definido como $\{ x \mid x \notin A \}$

Si A y B son conjuntos, el complemento relativo de A respecto a B es $B \setminus A = \{ x \mid x \in B \text{ y } x \notin A \}$

Ejemplo.- Si $U = \{0,1,2,3,4,5,6,7,8\}$

$A = \{1,4,6,7\}$

$$B = \{1,2,4,5,7\}$$

$$C = \{2,5\}$$

entonces

$$A^c = \{0,2,3,5,8\}$$

$$B^c = \{0,3,6,8\}$$

$$C^c = \{0,1,3,4,6,7,8\}$$

y el complemento relativo $\alpha \setminus \beta$ está dado por la siguiente tabla:

$\alpha \setminus \beta$	A	B	C
A	$\emptyset$	$\{6\}$	A
B	C	$\emptyset$	$\{1,4,7\}$
C	C	$\emptyset$	$\emptyset$

3.- COMBINACION DE CONJUNTOS

Partiendo de conjuntos dados, construiremos nuevos conjuntos.- Nuestro objetivo es llegar a un álgebra de conjuntos.

Definición 3.1

Sean A y B conjuntos cualesquiera.

La unión de A y B es $A \cup B = \{x \mid x \in A \text{ ó } x \in B \text{ o ambas cosas}\}$.

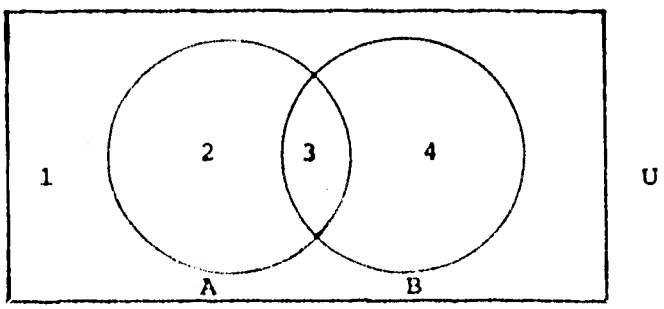
La diferencia simétrica de A y B es $A \Delta B = \{x \mid x \in A \text{ ó } x \in B \text{ pero no ambas}\}$.

La intersección de A y B es $A \cap B = \{x \mid x \in A \text{ y } x \in B\}$.

Ejemplo.-

En la siguiente tabla, se muestra la unión, la diferencia simétrica, la intersección y el complemento relativo de dos conjuntos dado un conjunto universal. Enseguida se dibujan los llamados diagramas de Venn para visualizar mejor estas relaciones.

conjunto	región	conjunto	región
U	1,2,3,4	$A \cup B$	2,3,4
A	2,3	$A \Delta B$	2,4
B	3,4	$A \cap B$	3
A^c	1,4	$A \setminus B$	2
B^c	1,2	$B \setminus A$	4



Aunque los diagramas de Venn son ayudas útiles para entender las operaciones básicas entre conjuntos, su uso es limitado, pues para cuatro o cinco conjuntos son muy complicados, y para números mayores se convierten en un maremagnum sin esperanza de aclaración. Afortunadamente en nuestro caso, no hay ninguna necesidad de considerar más de tres conjuntos a la vez para entender lo fundamental.

4.- ALGEBRA DE CONJUNTOS.

A medida que trabajamos con las operaciones de conjuntos, se van haciendo claras varias propiedades de estas operaciones. Por ejemplo, $\cup, \Delta$ y $\cap$ son conmutativas. Es decir:

$$A \cup B = B \cup A$$
$$A \Delta B = B \Delta A$$
$$A \cap B = B \cap A$$

Por otra parte, en general, $A \setminus B \neq B \setminus A$.

Algunas otras propiedades podemos verlas en la siguiente tabla:

$A \cup U = U$	$A \Delta U = A^c$	$A \cap U = A$
$A \cup A = A$	$A \Delta A = \emptyset$	$A \cap A = A$
$A \cup A^c = U$	$A \Delta A^c = U$	$A \cap A^c = \emptyset$
$A \cup \emptyset = A$	$A \Delta \emptyset = A$	$A \cap \emptyset = \emptyset$

$$A \cup B = (A \Delta B) \cup (A \cap B) = (A \Delta B) \Delta (A \cap B)$$
$$A \Delta B = (A \cup B) \setminus (A \cap B)$$
$$(A \cup B)^c = A^c \cap B^c$$
$$(A \cap B)^c = A^c \cup B^c$$
$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$
$$(A^c)^c = A$$

Vamos a sistematizar nuestro conocimiento de estas propiedades en forma de un álgebra de conjuntos.

Hay aquí un paralelo con el álgebra habitual de los números -- reales: aunque veremos que la analogía es imperfecta. Cuando - tratamos con números, las dos operaciones básicas son + y● y - satisfacen las siguientes propiedades:

Si a, b, c son números cualesquiera (reales)

,

Cierre:	A1	a + b es un número
	M1	a ● b es un número
Asociatividad:	A2	(a + b) + c = a + (b + c)
	M2	(a ● b) ● c = a ● (b ● c)
Identidad:	A3	Hay un único número 0 tal que para cual-- quier número a, a + 0 = 0 + a = a
	M3	Hay un único número 1, tal que para cual quier número a, a●1 = 1●a = a
Inverso:	A4	Para cada número a, hay un número único — a, tal que a + (-a) = (-a) + a = 0
	M4	Para cada número a ≠ 0, hay un número úni co a^{-1} tal que a●a^{-1} = a^{-1}●a = 1
Conmutatividad:	A5	a + b = b + a
	M5	a ● b = b ● a
Distributividad:	D1	a● (b + c) = a ● b + a ● c
	D2	(a + b) ● c = a ● c + b ● c

La analogía entre álgebra ordinaria y álgebra de conjuntos es-casi completa, si usamos:

$\triangle$ en lugar de +

y

$\cap$ en lugar de $\bullet$

Todas las propiedades excepto la M4 se verifican.

El "número único 0" se hace entonces "el único conjunto $\emptyset$", el "número único - a" es el mismo "conjunto A" que se trate, el - "número único 1"se convierte en "el único conjunto U".

Una diferencia fundamental entre el álgebra de conjuntos y el-álgebra ordinaria es que mientras que hay infinitos números, - puede ser que haya tan solo un número finito de conjuntos en - un álgebra de conjuntos.

Ejemplo.-

Sea $U = \{a, b, c\}$

Los subconjuntos de U son entonces ocho, a saber:

$\emptyset$	$D = \{a, b\}$
$A = \{a\}$	$E = \{a, c\}$
$B = \{b\}$	$F = \{b, c\}$
$C = \{c\}$	U

Las siguientes tablas muestran cuál es el álgebra de conjuntos basada en este conjunto U.

$\alpha \triangle \beta$

α \ β	$\emptyset$	A	B	C	D	E	F	U
$\emptyset$	$\emptyset$	A	B	C	D	E	F	U
A	A	$\emptyset$	D	E	B	C	U	F
B	B	D	$\emptyset$	F	A	U	C	E
C	C	E	F	$\emptyset$	U	A	B	D
D	D	B	A	U	$\emptyset$	F	E	C
E	E	C	U	A	F	$\emptyset$	D	B
F	F	U	C	B	E	D	$\emptyset$	A
U	U	F	E	D	C	B	A	$\emptyset$

$\alpha \cap \beta$

α \ β	$\emptyset$	A	B	C	D	E	F	U
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
A	$\emptyset$	A	$\emptyset$	$\emptyset$	A	A	$\emptyset$	A
B	$\emptyset$	$\emptyset$	B	$\emptyset$	B	$\emptyset$	B	B
C	$\emptyset$	$\emptyset$	$\emptyset$	C	$\emptyset$	C	C	C
D	$\emptyset$	A	B	$\emptyset$	D	A	B	D
E	$\emptyset$	A	$\emptyset$	C	A	E	C	E
F	$\emptyset$	$\emptyset$	B	C	B	C	F	F
U	$\emptyset$	A	B	C	D	E	F	U

En este ejemplo estamos haciendo uso de un resultado que probaremos enseguida, a saber, La determinación del número de objetos en un álgebra de conjuntos.

Como estos objetos son los subconjuntos de un conjunto universal dado, lo que realmente estamos determinando es el número de subconjuntos de un conjunto universal dado. Para el objetivo de este trabajo nos restringiremos solamente al caso finito.

Si $A = \{x_1, \ldots\ldots x_n\}$ es un conjunto de n elementos y $B \subseteq A$, entonces todo elemento $x_i \in A$ está en B o no está en B. Si definimos

$$Y_i = \begin{cases} 1 & \text{si } x_i \in B \\ 0 & \text{si } x_i \notin B \end{cases}$$

Entonces el conjunto B está caracterizado por una sucesión particular finita de ceros y unos $Y_1, \ldots, Y_n$

El número de subconjuntos B que pueden formarse es pues exactamente el número de tales sucesiones. Como hay dos posibilidades para cada Y_i y hay n Y_i - s, el número de tales sucesiones es el número de ordenaciones con repetición de dos elementos tomados de n en n, 2^n. De donde tenemos el siguiente resultado.

Teorema 4.1

Un conjunto de exactamente n elementos tiene exactamente 2^n subconjuntos.

Ejemplo.-

Sea $A = \{a, b, c,\}$ y supongamos la correspondencia

$$a \longrightarrow Y_1$$

$$b \longrightarrow Y_2$$

$$c \longrightarrow Y_3$$

La siguiente tabla muestra las sucesiones y los correspondientes subconjuntos de A.

Y_1	Y_2	Y_3	subconjunto
0	0	0	$\emptyset$
0	0	1	$\{c\}$
0	1	0	$\{b\}$
0	1	1	$\{b, c\}$
1	0	0	$\{a\}$
1	0	1	$\{a, c\}$
1	1	0	$\{a, b\}$
1	1	1	A

5.- PRODUCTO, RELACIONES Y MAPEOS

Definición 5.1

Sean A y B dos conjuntos.

El conjunto producto A x B se define por $A \times B = \{\langle a, b\rangle \mid a \in A \text{ y } b \in B\}$.

El símbolo $\langle a, b \rangle$ se denomina par ordenado, donde el término "ordenado" implica que distinguimos entre el primero y el segundo elemento.

Más generalmente:

Si $A_1, A_2, \ldots\ldots, A_n$ son conjuntos, entonces el conjunto - producto está definido como

$$A_1 \times A_2 \times \ldots \times A_n = \left\{ \langle a_1, a_2, \ldots, a_n \rangle \mid a_1 \in A, a_2 \in A_2, \ldots a_n \in A_n \right\}$$

El símbolo $\langle a_1, a_2, \ldots\ldots, a_n \rangle$ se llama n-ada ordenada.

Debe notarse que la operación producto no es conmutativa. Es - decir, en general $A \times B \neq B \times A$.

Analicemos ahora los subconjuntos de un conjunto producto, o - sea las relaciones.

Definición 5.2

Una relación es un subconjunto de un conjunto producto.

En particular, una relación n-aria es un subconjunto de - un conjunto producto de n conjuntos $A_1, A_2, \ldots\ldots, A_n$.

Ejemplo.

Sea $A = \{1, 2, 3\}$. Entonces $A \times A$ tiene 9 elementos.

La relación (binaria) menor que es entonces el subconjunto
$M = \{\langle 1,2 \rangle , \langle 1,3 \rangle , \langle 2,3 \rangle\}$.

La relación (binaria) igual que es el subconjunto

$$I = \{\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,3\rangle\}$$

Centraremos nuestra atención en las relaciones de orden 2; o - sea en las relaciones binarias ya que de ahí surge el importante concepto de función.

Definición 5.3

Una relación binaria R es una función o mapeo si siempre- que $\langle a,b\rangle \in R$ y $\langle a,c\rangle \in R$, entonces $b = c$.

Sin perder de vista el papel jugado por la relación de orden 2 en la definición anterior, podemos definir lo que es una fun-- ción si nuestra relación es n-aria, de la siguiente manera:

Una relación n-aria R_n es una función o mapeo si siempre- que $\langle a_1, a_2, \ldots, a_{n-1}, b\rangle \in R_n$ y

$\langle a_1, a_2, \ldots, a_{n-1}, c\rangle \in R_n$, entonces $b = c$.

En particular, si $R \subseteq A_1 \times A_2 \times \ldots \times A_{n-1} \times A_n$ es un mapeo, es un mapeo de $A_1 \times A_2 \times \ldots \times A_{n-1}$ en A_n.

Diremos que el mapeo R es sobre, si para cada $b \in A_n$ existen - elementos $a_i \in A_i$, $i = 1, \ldots, n-1$, tales que $\langle a_i, \ldots, a_{n-1}, b\rangle \in R$.

Más adelante tendremos necesidad de mapeos en el conjunto de - los enteros positivos (longitud de palabras, número de paréntesis, etc.), en el conjunto $\{V,F\}$ (interpretadas como verdadero y falso) y también en un conjunto de gráficas (el "árbol" - de una fórmula).

Definición 5.4

Si $R \subseteq A \times B$ es una relación binaria, entonces su relación inversa es $R^{-1} \subseteq B \times A$, definida por $\langle a,b\rangle \in R^{-1}$, si y sólo si $\langle b,a\rangle \in R$.

Nótese que la inversa de una función no necesariamente es una función. Nótese también que $(R^{-1})^{-1} = R$.

Definición 5.5

Si R y R^{-1} resultan ser ambas funciones, entonces R y R^{-1} se dice que son funciones inyectivas o uno-uno.

El término uno-uno nace del hecho de que un mapeo tal envía dos elementos distintos en dos elementos distintos. Tales mapeos los utilizaremos más adelante, por ejemplo en la recuperación de información. Si hemos asignado números a expedientes en un banco de datos, (es decir, si hemos definido un mapeo del conjunto de expedientes en el conjunto de números), entonces para poder localizar los expedientes usando los números, tendríamos que haber asignado números distintos a expedientes distintos, o sea ,el mapeo habría de ser uno-uno.

6.- LA RELACION SUBCONJUNTO.

Examinamos ahora con más detalle la única relación que hemos usado entre conjuntos, la de subconjunto. En el sentido definido en la sección anterior, esta relación es la siguiente:

Si U es nuestro conjunto universal y V el conjunto de todos los subconjuntos de U, entonces $A \subseteq B$ es una relación en $V \times V$.

Esta relación tiene las siguientes propiedades en común con la

relación "menor o igual que" ($\leqslant$) definida para números.

	números	conjuntos
Reflexiva:	$n \leqslant n$	$A \subseteq A$
Antisimétrica	si $n \leqslant m$ y $m \leqslant n$ entonces $n = m$	si $A \subseteq B$ y $B \subseteq A$ entonces $A = B$
Transitiva:	si $n \leqslant m$ y $m \leqslant p$ entonces $n \leqslant p$	Si $A \subseteq B$ y $B \subseteq C$ entonces $A \subseteq C$

Toda relación con estas propiedades se denomina un orden parcial.

Así pues, la relación subconjunto es un orden parcial sobre el conjunto de todos los subconjuntos de U.

Sin embargo, una diferencia notable entre las relaciones $\leqslant$ y $\subseteq$ es que:

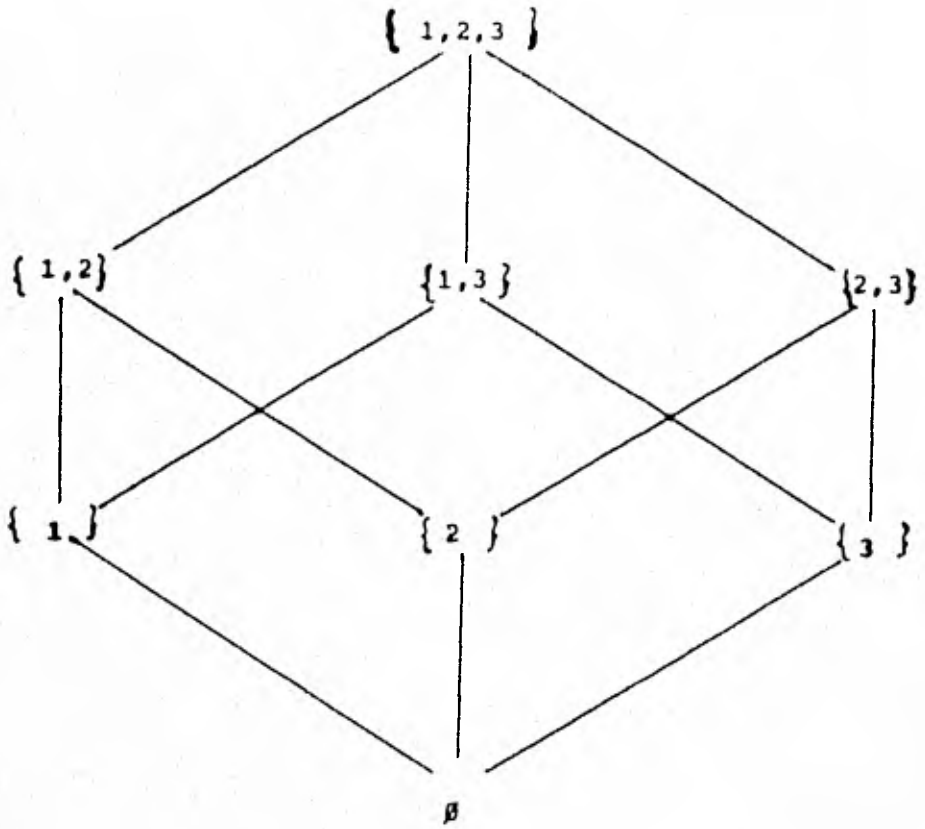
Dados dos números cualesquiera m y n, se tiene $m \leqslant n$ ó $n \leqslant m$.

Pero dados, por ejemplo, los conjuntos $A = \{1,2\}$ y $B = \{2,3\}$ ninguno es subconjunto del otro.

Las relaciones subconjunto existentes en un conjunto universal dado podemos expresarlas mediante un dibujo llamado reticulado de subconjuntos ó diagrama de Hasse. En un tal diagrama, $A \subseteq B$ si y sólo si A se encuentra debajo de B y está conectado con B por un segmento o segmentos en dirección ascendente.

Ejemplo.

El siguiente diagrama muestra el reticulado de los subconjuntos del conjunto $\{1,2,3\}$.


{ 1,2,3 }
{ 1,2}
{1,3 }
{2,3}
{ 1 }
{ 2 }
{ 3 }
ø

b) ALGEBRAS BOOLEANAS.

1. Definición y comentarios
2. Dualidad y otras propiedades
3. Funciones booleanas
4. De los conjuntos a la lógica
5. Aplicación booleana a circuitos
6. Diseño lógico de un circuito sumador

1.- DEFINICION Y COMENTARIOS

Definición 1.1

Un <u>álgebra booleana</u> es un sistema algebraico constituido por un conjunto S junto con dos operaciones # (como adición) y $\bullet$ (como multiplicación) definidas sobre S, tales que:

1) Se verifican las reglas:

A_1 a A_5

M_1 a M_3

M_5

D_1 y D_2

(ver el apartado a), sección 4).

y

2) Todo elemento es un idempotente, es decir, si $a \in S$, entonces $a \bullet a = a$.

Resulta claro ahora, dada esta definición, que toda álgebra de conjunto es un álgebra booleana, con Δ como adición y $\cap$ como multiplicación. Sin embargo, en el caso recíproco, hacemos notar que toda álgebra booleana, si es finita, no es más que el álgebra de todos los subconjuntos de algún cierto conjunto universal dado, pero no así en caso de ser infinita, pues algunas de ellas forman álgebras de conjuntos más complejas.

Hemos evitado deliberadamente usar " + " como el símbolo de la adición porque por lo general se asocia con la operación unión (U) de las álgebras de conjuntos.

Definimos ahora en términos de + y ● las álgebras booleanas. - Esta y la anterior definición son equivalentes.

Definición 1.2

Un álgebra booleana es un sistema algebraico consistente- en un conjunto S junto con dos operaciones + y ● defini-- das sobre S, tales que para cualesquiera elementos - - -- a, b, c ∈ S, se verifican las siguientes propiedades o - axiomas:

A1. $a + b \in S$

M1. $a \bullet b \in S$

A2. $a + (b + c) = (a + b) + c$

M2 $a \bullet (b \bullet c) = (a \bullet b) \bullet c$

A3. $\exists! \, o \in S \ni \forall \, a \in S: \; a + 0 = 0 + a = a$

M3. $\exists! \, 1 \in S \ni \forall \, a \in S: \; a \bullet 1 = 1 \bullet a = a$

A5. $a + b = b + a$

M5. $a \bullet b = b \bullet a$

D1. $a \bullet (b + c) = a \bullet b + a \bullet c$

D2. $(a + b) \bullet c = a \bullet c + b \bullet c$

D3. $a + (b \bullet c) = (a + b) \bullet (a + c)$

D4. $(a \bullet b) + c = (a + c) \bullet (b + c)$

C1. $\forall \, a \in S \; \exists! \, a' \in S \ni a + a' = 1 \text{ y } a \bullet a' = 0$

Donde en la notación anterior:

$\exists!$ quiere decir "hay un único"
$\forall$ " " "para todo"
$\ni$ " " "tal que"
: " " "se cumple lo siguiente"

2.- DUALIDAD Y OTRAS PROPIEDADES.

Trabajaremos con la segunda definición de las que dimos para - álgebras booleanas.

Notemos que todos los axiomas dados, con excepción de A3, M3 y C1, se presentan en pares, con la propiedad de que, si se in-- tercambian la adición y la multiplicación, el único efecto es- el de intercambiar los miembros del par, —no se crean nuevas- fórmulas.

D1 se convierte en D3 y viceversa; lo mismo D2 con D4.

En cambio, si intercambiamos las operaciones en A3, M3 y C1 sí se originan nuevas fórmulas. Sin embargo, si el intercambio - lo hacemos también con el 0 y el 1, A3 se convierte en M3 y vi ceversa y en C1 se intercambian sus dos fórmulas. Tenemos - - pues una importante dualidad en las álgebras booleanas.

PRINCIPIO DE DUALIDAD

> Si una fórmula dada es deducible de los axiomas de un álgebra booleana, entonces la fórmula obtenida al intercambiar + y ● y los elementos 0 y 1 en la fórmula dada, es - también deducible.

Este principio se sigue de los comentarios hechos antes sobre- la dualidad de los axiomas, y del hecho de que cada prueba o -

deducción consiste en una sucesión de fórmulas que son axiomas o son derivadas de los axiomas.

A continuación veremos unas pocas propiedades con pruebas in-- formales aplicando el principio anterior y que utilizaremos -- después.

Suponemos que $\mathbb{B} = \langle S, +, \bullet \rangle$ es un álgebra booleana y que - todos los elementos que aparecen son miembros de S.

Teorema 2.1

$$\forall a \in S: \quad \underline{a + a = a} \quad y \quad \underline{a \bullet a = a}$$

Prueba,

$a + a = (a + a) (1)$ M3

$= (a + a) (a + a')$ C1

$= a + (a \bullet a')$ D3

$= a + 0$ C1

$= a$ A3

Q.E.D.

La prueba para $a \bullet a = a$ se obtiene automáticamente por dualidad.

$a \bullet a = a \bullet a + 0$ A3

$= (a \bullet a) + (a \bullet a')$ C1

$= a (a + a')$ D1

$= a \bullet 1$ C1

$= a$ M3

Teorema 2.2

$$\forall\, a \in S:\ \underline{a + 1 = 1}\ \text{ y }\ \underline{a \bullet 0 = 0}$$

Prueba.

$a + 1 = 1\ (a + 1)$ M3

$= (a + a')\ (a + 1)$ C1

$= a + (a' \bullet 1)$ D3

$= a + a'$ M3

$= 1$ ..... C1.

Q.E.D.

La prueba para $a \bullet 0 = 0$ se sigue de ésta por dualidad.

Teorema 2.3 LEYES DE ABSORCION

$$\forall\, a, b \in S:\ \underline{a + (a\ b) = a}\ \text{ y }\ \underline{a \bullet (a + b) = a}$$

Prueba.

$a + (a\ b) = (a \bullet 1) + (a \bullet b)$ M3

$= a\ (1 + b)$ D1

$= a \bullet 1$ ..... A5, Teo 2.2

$= a$ M3

Q.E.D.

La prueba para $a\ (\ a + b\ ) = a$ se sigue de ésta por dualidad.

Teorema 2.4 LEYES DE DEMORGAN

$$\forall a, b \in S: (a \bullet b)' = a' + b' \text{ y } (a + b)' = a' \bullet b'$$

Prueba,

Por C1 sabemos que $(a \bullet b)(a \bullet b)' = 0$ y $(a \bullet b) + (a \bullet b)' = 1$ y que para $a \bullet b$, $(a \bullet b)'$ es el único elemento para el que estas afirmaciones son válidas.

Entonces, si podemos mostrar que $a' + b'$ en el lugar de $(a.b)'$ satisface estas mismas ecuaciones, tenemos la primera fórmula-del Teorema. La segunda se sigue por dualidad. Examinamos -- pues $(a \bullet b)(a' + b')$ y $(a \bullet b)+(a' + b')$.

$(a \bullet b)(a' + b') = a \bullet b \bullet a' + a \bullet b \bullet b'$ D1, M2

$= 0 \bullet b + a \bullet 0$ M2, M5, C1

$= 0 + 0$ ..... Teo 2.2

$= 0$ A3

$(a \bullet b)+(a' + b') = (a + a' + b')(b + a' + b')$ D4

$= (1 + b')(1 + a')$ A2, A5, C1

$= 1 \bullet 1$ ..... Teo 2.2

$= 1$ M3

Por tanto, tenemos que $(a \bullet b)' = a' + b'$

Q.E.D.

Teorema 2.5

$$0' = 1 \quad y \quad 1' = 0$$

Prueba.

Sabemos por el Teo. 2.2 que $1 + 0 = 1$ y $1 \bullet 0 = 0$, pero éstas son las ecuaciones de C1, y por la unicidad allí postulada, $1' = 0$ y $0' = 1$

Q.E.D.

Teorema 2.6

Si un álgebra booleana contiene al menos dos elementos distintos, entonces $0 \neq 1$.

Prueba.

Supongamos que $\exists$ $\mathbb{B} = \langle S, +, \bullet \rangle$ álgebra booleana con almenos dos elementos distintos en la que $0 = 1$. $\Rightarrow$ $\exists$ $a \in S \ni a \neq 0$.

Por un lado tenemos, por M3: $a = a \bullet 1$

Por otro, $1 = 0 \Rightarrow$ $a \bullet 1 = a \bullet 0$ y $a \bullet 0 = 0$ por Teo. 2.2

$\therefore$ $a = 0$ $\overset{\nabla}{\circ}$ De donde $0 \neq 1$.

Q.E.D

El símbolo $\overset{\nabla}{\circ}$ denota "contradicción".

Corolario 1.

Ningún álgebra booleana tiene exactamente tres elementos.

Prueba.

Supongamos que hay tres elementos distintos en un álgebra booleana, a saber, 0, 1 y a. (Sabemos que 0 y 1 están ambos presentes y son distintos por el Teo. 2.6).

Consideremos a'. De la ecuación a + a' = 1 y a ● a' = 0 se sigue que a' no puede ser ninguno de los elementos 0,- a ó 1. De donde debe existir, según C1, un cuarto elemento.

Q.E.D.

Lema.- (a')' = a $\forall$ a $\in$ S . (por la unicidad postulada en C1 - pues a' $\in$ S).

Corolario 2.

Ningún álgebra booleana finita con más de un elemento tiene un número impar de elementos.

Prueba.

De las ecuaciones a + a' = 1 y a ● a' = 0 y de los Teos.- 2.6 y 2.1 se sigue que para cada a, a' es distinta de a.- También se sigue que si a $\neq$ b entonces a' $\neq$ b', pues si - a' = b' = c entonces c' = (a')' y c' = (b')' luego - - - a = (a')' = (b')' = b. Por tanto los elementos de un álgebra booleana con más de un elemento deben ocurrir en pares distintos (0,1), (a,a'), (b,b'),.... cuya existencia se desprende de C1.

Q.E.D.

Como toda álgebra de conjuntos es un álgebra booleana, el orden parcial que es la relación subconjunto debe tener validez, al - menos para algunas álgebras booleanas. Veremos que en realidad es válido para todas ellas.

Definición 2.1

Sean x y y elementos de un álgebra booleana. Decimos que x es menor o igual que y ($x \leqslant y$) si y sólo si $x + y = y$.

Teorema 2.7

$\leqslant$ es un orden parcial.

Prueba.

- Por el Teo. 2.1 $x + x = x$ $\therefore x \leqslant x$
- Si $x \leqslant y$ entonces $x + y = y$

  y si $y \leqslant x$ entonces $y + x = x$

  pero $x + y = y + x$ $\therefore x = y$
- Si $x \leqslant y$ y $y \leqslant z$ entonces $x + y = y$ y $y + z = z$

  de donde $x + z = x + (y + z) = (x + y) + z =$

  $y + z = z$ $\therefore x \leqslant z$.

Q.E.D.

Teorema 2.8

Sean x, y, z elementos de un álgebra booleana.

Entonces el orden parcial tiene las siguientes propiedades:

i) Si $x \leqslant y$ y $x \leqslant z$, entonces $x \leqslant y \cdot z$

ii) Si $x \leqslant y$, entonces $x \leqslant y + z$ para cualquier z.

iii) Si $x \leqslant y$, entonces $x \cdot z \leqslant y$ para cualquier z.

iv) $x \leqslant y \iff y' \leqslant x'$

El símbolo " $\Leftrightarrow$ " significa "si y sólo si".

Pruebas.

i) $x + y = y \underline{y}\ x + z = z$. Luego por D) $x + y\ z =$ $(x + y)\ (x + z) = yz \therefore x \leq y\ z$.

ii) Si $x + y = y$, entonces $x + (y + z) = (x+y) + z = x + z$ $\therefore x \leq y + z$.

iii) Por las leyes de absorción (Teo. 2.3) $x\ z + x = x$. Luego $x\ z \leq x \underline{y}$ como $x \leq y$, por transitividad $x\ z \leq y$.

iv) Supongamos $x \leq y$. Entonces $x + y = y$, de donde $y' = (x + y)'$. Luego $y' + x' = (x + y)' + x' =$ $= \left[(x + y) \bullet x\right]' = x'$ por las leyes de absorción.

El recíproco se sigue del Lema $(x')' = x$ dado antes.

Q.E.D.

3.- FUNCIONES BOOLEANAS

Definición 3.1

Sean $x_1, \ldots\ldots, x_n$ variables cuyos valores se encuentran en un álgebra booleana. Un mapeo f del álgebra booleana en sí misma:

$$f : \mathbb{B} = \langle S, +, \bullet \rangle \longrightarrow \mathbb{B}$$

es una <u>función booleana de n variables</u>, si puede construirse de acuerdo con las siguientes cinco reglas:

1.- Para valores cualesquiera de $x_1, \ldots, x_n$, $f(x_1, \ldots, x_n) = a$, donde $a \in S$ es fija, entonces f es una función booleana. Esta es la función <u>constante.</u>

2.- Si para valores cualesquiera de $x_1, \ldots, x_n$, $f(x_1, \ldots, x_n) = x_i$ para algún $i = 1, \ldots, n$, entonces f es una función booleana. Esta es una función <u>proyección.</u>

3.- Si f es una función booleana, entonces g, definida por $g(x_1, \ldots, x_n) = (f(x_1, \ldots, x_n))'$ para cualesquiera $x_1, \ldots, x_n$, es una función booleana.

4.- Si f y g son funciones booleanas, entonces h y k definidas por:

$$h(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) + g(x_1, \ldots, x_n)$$

$$k(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) \bullet g(x_1, \ldots, x_n)$$

para $x_1, \ldots, x_n$ cualesquiera, son funciones booleanas.

5.- Cualquier función que pueda construirse por un número <u>finito</u> de aplicaciones de las reglas anteriores, y sólo funciones de tal tipo, es una función booleana.

En resumen, vemos que una función booleana es cualquier función que puede construirse partiendo de las funciones constantes y las funciones proyección mediante un número finito de usos de $'$, $+$ y $\bullet$.

A causa de las relaciones existentes entre las operaciones, es-posible que una función booleana tome diversas formas.

Ejemplo:

Si $f(x, y) = x' y'$

y $g(x, y) = (x + y)'$ entonces por De Morgan sabemos que-

f y g son la misma función.

Para determinar mejor si dos expresiones representan o no la - misma función booleana es deseable tener una forma estándar para representar las funciones. En el teorema que sigue se pre-- senta tal forma estándar llamada <u>forma canónica.</u>

Teorema 3.1

Si f es una función booleana de una variable, entonces para todos los valores de x, $f(x) = f(1) \bullet x + f(0) \bullet x'$.

Este es el caso de una sola variable x. De manera análoga tenemos que:

Si f es una función booleana de dos variables, entonces - para todos los valores de x y de y, $f(x,y) = f(1,1) \bullet x \bullet y + f(1,0) \bullet x \bullet y' + f(0,1) \bullet x' \bullet y + f(0,0) \bullet x' \bullet y'$.

Y en general,

Si f es una función booleana de n variables, entonces para todos los valores de $x_1, \ldots, x_n$,

$$f(x_1, \ldots, x_n) = \sum f(e_1, \ldots, e_n)\, x_1^{e_1} \cdot x_2^{e_2} \ldots x_n^{e_n}$$

donde las e_i toman los valores 0 ó 1 y

$$x_i^{e_i} = \begin{cases} x_i \text{ si } e_i = 1 \\ x_i' \text{ si } e_i = 0 \end{cases}$$

Nótese que la sumatoria desarrolla 2^n términos.

Esta forma canónina se conoce como una SUMA DE PRODUCTOS ó forma normal disyuntiva o FORMA NORMAL MINTERM. En formal dual, - hay un PRODUCTO DE SUMAS o forma normal conjuntiva o FORMA NORMAL MAXTERM, que discutiremos en el capítulo siguiente.

Ejemplo.-

En un álgebra booleana con 4 elementos 0, 1, a, a', consi deremos las funciones f(x) = x + x • a y g(x,y) = x' y + x y' + y'. Observando la siguiente tabla podemos trans-- formar f y g a su forma canónica, considerando los valo-- res que toman en 0 y 1.

x	f (x)
0	a
1	1
a	a
a'	1

x \ y	0	1	a	a'
0	1	1	1	1
1	1	0	a'	a
a	1	a'	a'	1
a'	1	a	1	a

g(x,y)

Forma normal Minterm:

f(x) = f(1) • x + f(0) . x'

$$g\ (x,y) = 0 \bullet yx + 1 \bullet xy' + 1 \bullet x'y + 1 \bullet x'y' \ .$$

De la forma normal disyuntiva se desprende, de modo evidente, que para cualquier álgebra booleana los valores de cualquier función booleana f de n variables están completamente determinados por sus valores en los puntos $\langle x_1, \ldots, x_n \rangle$, donde cada una de las x_i tiene el valor 0 ó el 1. Nos proporciona esto una bella interpretación geométrica de las funciones booleanas.

4.- DE LOS CONJUNTOS A LA LOGICA

Veremos en forma breve el nexo que mencionamos entre la teoría de conjuntos y la Lógica, que nos proporciona el álgebra booleana.

A un extremo tenemos un álgebra de conjuntos de dos elementos basada, digamos, en el conjunto $A = \{a\}$ con sus dos subconjuntos $\emptyset$ y A. Las tres tablas de operaciones para ella aparecen en la Tabla 4.1.

$\alpha \Delta \beta$

α \ β	$\emptyset$	A
$\emptyset$	$\emptyset$	A
A	A	$\emptyset$

$\alpha \cup \beta$

α \ β	$\emptyset$	A
$\emptyset$	$\emptyset$	A
A	A	A

$\alpha \cap \beta$

α \ β	$\emptyset$	A
$\emptyset$	$\emptyset$	$\emptyset$
A	$\emptyset$	A

Tabla 4.1. Operaciones en álgebra de conjuntos de dos elementos.

Interpretando ésta como un álgebra booleana, encontramos que los dos elementos $\emptyset$ y A deben corresponder al 0 y 1 que están presentes en toda álgebra booleana.

$\emptyset$ corresponde al 0 y A corresponde al 1. Las operaciones para

un álgebra booleana abstracta sobre dos elementos están dadas en la Tabla 4.2

$\alpha \# \beta$

α \ β	0	1
0	0	1
1	1	0

$\alpha + \beta$

α \ β	0	1
0	0	1
1	1	1

$\alpha \bullet \beta$

α \ β	0	1
0	0	0
1	0	1

Tabla 4.2. Operaciones en álgebra booleana de dos elementos.

La Lógica más a menudo estudiada es una lógica de dos valores, (es la más estudiada por razones que su aplicación justifica). Una proposición puede ser verdadera (V) o falsa (F), pero nada más. Los símbolos operatorios se corresponden de la manera siguiente:

álgebra booleana		Lógica
#	⟶	|
+	⟶	$\vee$
•	⟶	$\wedge$

Las tablas operatorias se dan en la tabla 4.3

$\alpha \nmid \beta$

α \ β	F	V
F	F	V
V	V	F

$\alpha \vee \beta$

α \ β	F	V
F	F	V
V	V	V

$\alpha \wedge \beta$

α \ β	F	V
F	F	F
V	F	V

Tabla 4.3 Operaciones en el cálculo proposicional.

En la serie de tablas anteriores estoy tomando:

En conjuntos.- $\alpha \Delta \beta \doteq \{ x \mid x \in \alpha \text{ ó } x \in \beta, \text{ pero no ambas} \}$

En álgebra bolleana.- $\alpha \nmid \beta \doteq (\alpha' \bullet \beta) + (\alpha \bullet \beta')$

En Lógica.- $\alpha \nmid \beta \doteq (\alpha \vee \beta) \wedge \sim (\alpha \wedge \beta)$

donde los conectivos $\vee, \wedge, \sim$ los discutiremos en el capítulo siguiente.

La correspondencia total entre símbolos es entonces:

conjuntos		álgebra booleana		Lógica
Δ	$\longrightarrow$	$\nmid$	$\longrightarrow$	$\nmid$
$\cup$	$\longrightarrow$	$+$	$\longrightarrow$	$\vee$
$\cap$	$\longrightarrow$	$\bullet$	$\longrightarrow$	$\wedge$

5.- APLICACION BOOLEANA A CIRCUITOS

Como una aplicación física inmediata del álgebra booleana consideraremos el diseño de circuitos de distribución.

Podemos asociar a cada variable con un interruptor o válvula - en una línea por la que puede pasar electricidad, agua, información o cualquier otra magnitud. Físicamente, este interruptor puede ser una válvula en un sistema de conducción de aguas, un interruptor eléctrico, un transistor, un diodo, un despachador de un almacén, o cualquier otro mecanismo o persona capaz de permitir o prohibir el paso de alguna magnitud.

Por ejemplo en la figura 5.1 a), si hay una entrada en el punto a, entonces hay una salida en el punto b si y sólo si la -- válvula x está cerrada.

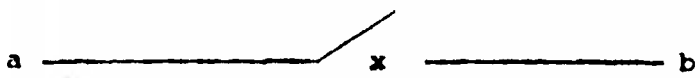


Fig. 5.1 a) x'

Podemos entonces representar varias funciones lógicas en términos de tales válvulas.

- Una variable "no complementada" la asociamos a una válvula que está normalmente abierta.
- Una variable "complementada" la asociamos a una válvula que está normalmente cerrada.

De modo que:

- Habrá una salida, si la variable es pasiva (válvula izquierda cerrada).
- No habrá salida, si la variable es activa (válvula - - abierta).

Así, en la figura 5.1b), si hay una entrada en el punto a, ha-

brá entonces una salida en el punto b, si y sólo si ambas válvulas x,y, están cerradas. Luego esto representa el término - xy.

Fig. 5.1b) x•y

Vamos, por último, a la figura 5.1 c)

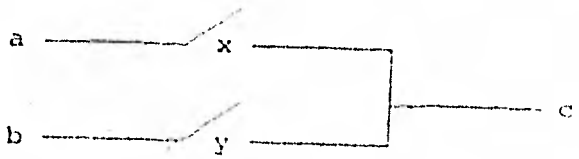


Fig. 5.c) x + y

Para que haya una salida en el punto c, dado que haya entradas en los puntos a y b, o la válvula x o la válvula y, o ambas, deben estar cerradas. De donde esta configuración representa la expresión x + y.

Con esto tenemos los circuitos básicos de distribución que representan las operaciones básicas de un álgebra booleana: - - ' , • , + .

Vayamos ahora a expresiones más complejas mediante ejemplos -- sencillos.

Ejemplo 5.1

Determinemos qué función booleana representa el circuito de la figura 5.2.

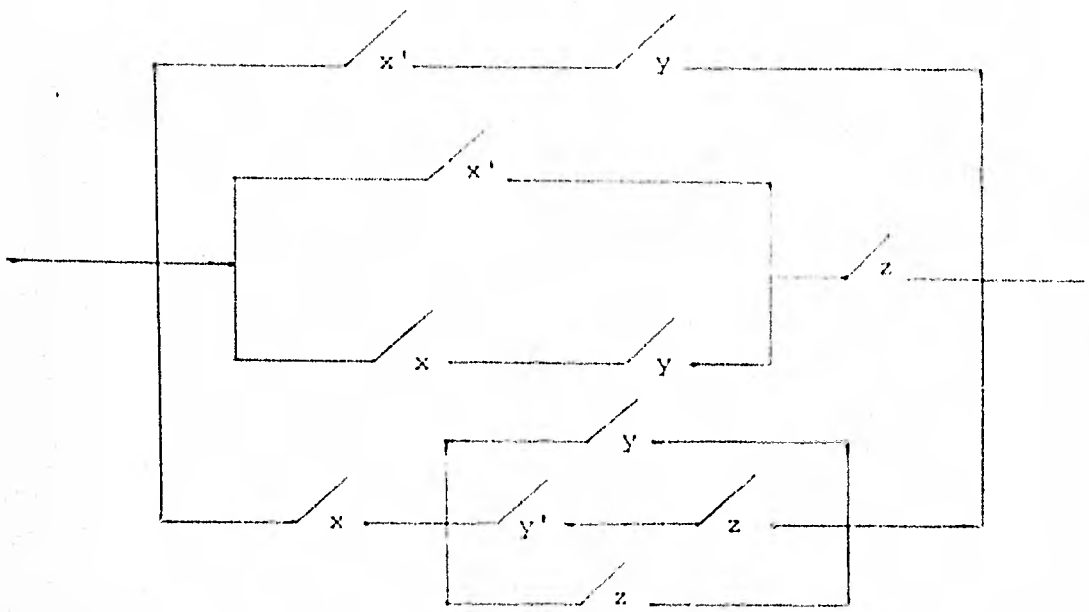


Fig. 5.2

(Recuérdese que la válvula x está normalmente abierta y la x'- normalmente cerrada).

- La línea superior en el circuito representa la expresión- x'y.

- La segunda línea, en la porción izquierda, tenemos el tipo de la figura 5.1 c), y por ende representa x' + xy. -- Luego, la línea en su totalidad, corresponde a la expre-- sión (x' + xy) z.

- Análogamente, la línea inferior se corresponde con la expresión x (y + y' z+ z).

El circuito total es de nuevo del tipo mostrado en la fig. - 5.1 c), de donde representa la expresión booleana:

$$x'y + (x' + xy)\ z + x\ (y + y'\ z + z)$$

Ejemplo 5.2

Analicemos el siguiente caso.

La expresión w' x' z + w xz' + xy se corresponde con la - figura 5.3

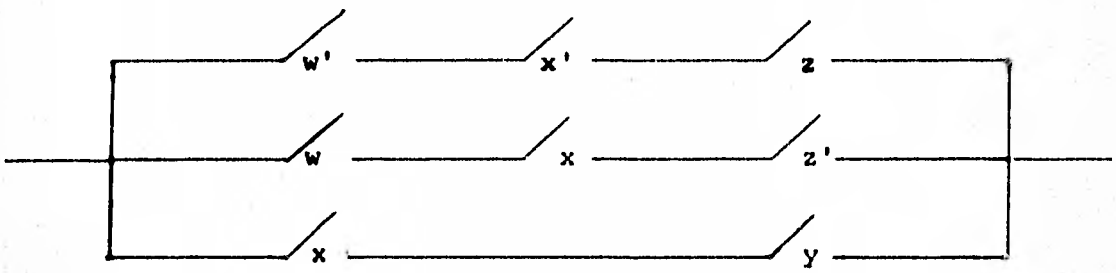


Fig. 5.3

Nótese que dos de las válvulas tienen como denominación x.

Podemos combinar estas dos válvulas obteniendo el circuito de- la Fig. 5.4, que tiene una válvula menos y es equivalente al - circuito de la Fig. 5.3. Esto corresponde a la factorización- de x.

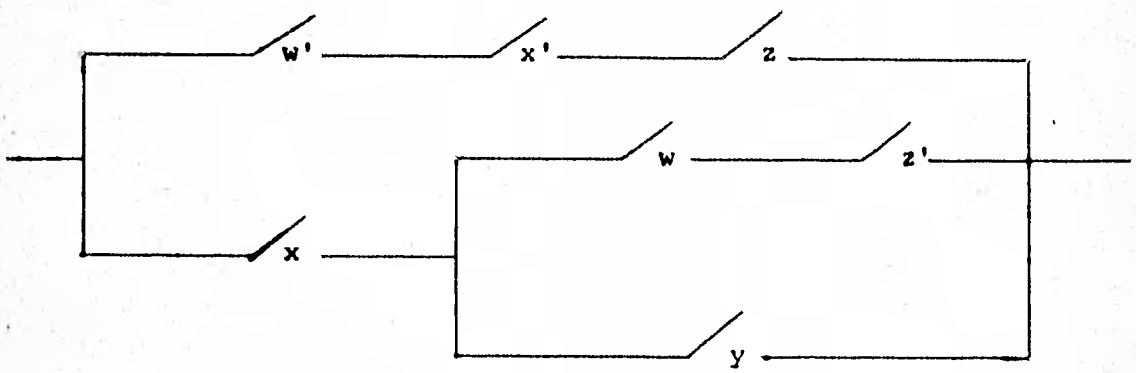
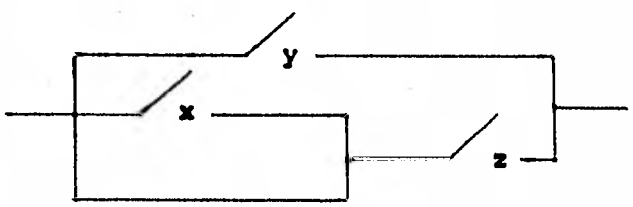


Fig. 5.4 Circuito equivalente al 5.3

A causa de la correspondencia entre las funciones booleanas y los circuitos sencillos de distribución, es posible aplicar técnicas de minimización del álgebra booleana — gráficas, tabulares o algebraicas — al problema de simplificar un circuito de distribución dado.

Así, nuestra expresión del ejemplo 5.1, $x'y + (x' + xy)\ z + x\ (y + y'\ z + z)$ la reducimos rápidamente en forma algebraica a la expresión $y + (x + 1)\ z$, que implica el siguiente circuito de distribución mucho más simple, que es entonces equivalente de la figura 5.2



Circuito Equivalente al 5.2

6.- DISEÑO LOGICO DE UN CIRCUITO SUMADOR

Veremos ahora un trabajo de aplicación de la matemática al diseño de un circuito sumador similar a los usados en las calculadoras electrónicas. Se enfatiza la importancia de la lógica matemática y la aritmética, así como la forma en que ambas se combinan para lograr el artefacto en cuestión.

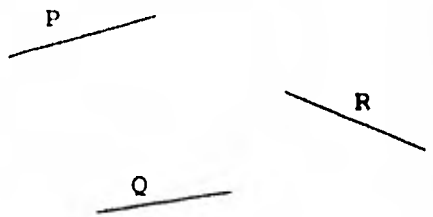
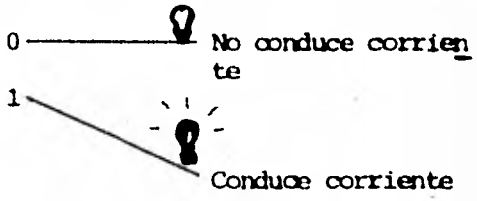
Empezaremos por observar la manera en que el simbolismo más elemental de la lógica matemática, ideado y desarrollado para estudiar la estructura de los lenguajes natural y matemático, se presta a una reinterpretación que desempeña una función determinante en el proceso de aplicación. Luego compararemos

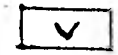
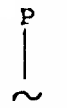
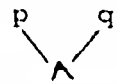
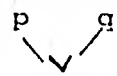
brevemente algunos sistemas de representación de números naturales, intentando poner en claro que la forma en que hacemos - operaciones aritméticas en el sistema decimal se puede exten-- der a otras notaciones posicionales y, por último, explicare-- mos la forma en que se integran los conceptos lógicos y aritmé ticos para el logro del circuito digital antes mencionado.

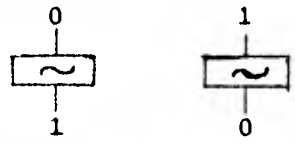
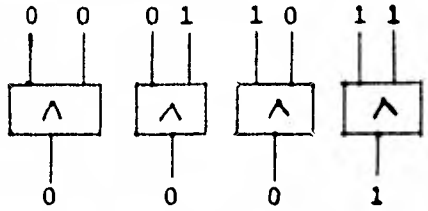
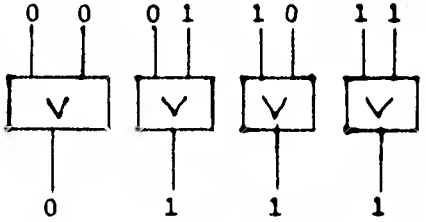
De los conceptos Lógicos, sólo utilizamos los más elementales- del cálculo proposicional y los presentaremos no rigurosamente, sino por medio de algunos usos que se les da y lo haremos en - la Tabla 6.0, organizada así:

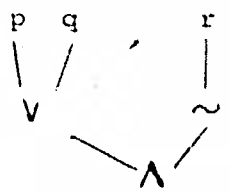
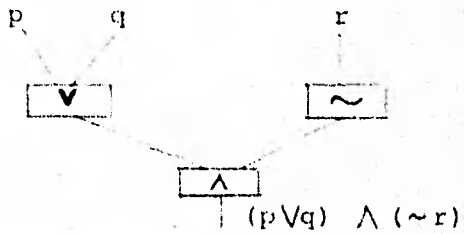
La columna central está dedicada al simbolismo correspondiente a los conceptos mencionados y tiene prioridad en la Tabla. En- la primera columna se dan ejemplos de construcciones, pertenecientes al español y a la matemática, similares a las que originaron el uso del simbolismo lógico. Con el transcurso del - tiempo la Lógica matemática se ha desarrollado y los puntos de vista han cambiado: en la actualidad en vez de considerar a -- los símbolos lógicos como meras abreviaturas de ciertas cons-- trucciones de otros lenguajes, conviene tratar a éstas como in terpretaciones de aquellos, entre muchas otras posibles. Preci samente en la tercera columna se presenta otra interpretación, una en la que a los símbolos lógicos se les asocian objetos de los comúnmente llamados "materiales", en contraste con las interpretaciones anteriores en que los objetos asociados son sim bólicos. Esto no tiene nada de raro, tan solo ilustra que la- Lógica Matemática se puede aplicar a la matemática misma o al- "mundo real", conviene notar que en este caso la aplicación -- "práctica" fue posterior y en mucho se debe a la aplicación -- "teórica".

TABLA 6.0.

INTERPRETACIONES USUALES	ALGUNOS SIMBOLOS USADOS EN LOGICA MATEMATICA	INTERPRETACION DE INTERES PARA NUESTRO TRABAJO
$\lim_{x \to xo} \frac{f(x) - f(xo)}{x - xo} = f'(xo)$ Guadalajara es bella 2 + 1 = 3 (Ejemplos de proposiciones; dos pertenecen al lenguaje matemático y la otra al español).	p Q R (Símbolos conocidos como letras proposicionales; originalmente se pensaron para representar proposiciones	 (a las letras proposicionales les corresponden alambres conductores de electricidad, que en el papel representamos con rayas).
FALSO VERDADERO (Usados para clasificar proposiciones en verdaderas y falsas)	0 1 (Se les conoce como valores de verdad)	 (Cuando no conduce corriente un alambre se dice que está en estado 0, en caso contrario se dice que está en estado 1)

INTERPRETACIONES USUALES	ALGUNOS SIMBOLOS USADOS EN LOGICA MATEMATICA	INTERPRETACION DE INTERES PARA NUESTRO TRABAJO
NO Y O (Se combinan con proposiciones para dar lugar a- proposiciones compuestas)	$\sim$ $\wedge$ $\vee$ (Llamados operadores lógicos). Se combinan con letras proposicionales para construir fórmulas proposicionales. Se llaman negación, conjunción y disyunción, respectivamente).	   (A los operadores les corresponden unos artefactos cuyo comportamiento describimos más adelante. Se conectan con alambres para - construir circuitos).
$2 + 1 \neq 5$ (abrevia: no $2 + 1 = 5$) $1 < 3 < 5$ (abrevia: $1 < 3$ y $3 < 5$) $2 \leq 3$ (abrevia: $2 < 3$ ó $2 = 3$) (Son ejemplos de proposiciones compuestas).	$\sim p$  $p \wedge q$  $p \vee q$  (Ejemplos de Fórmulas Proposicionales, escritas en forma - - usual y en forma de árbol).	  (Ejemplos de circuitos de los más sencillos).

INTERPRETACIONES USUALES	ALGUNOS SIMBOLOS USADOS DE LOGICA MATEMATICA	INTERPRETACION DE INTERES PARA NUESTRO TRABAJO
Cuando una proposición es falsa, su negación es verdadera y viceversa.	A | ~A 0 | 1 1 | 0	
La conjunción de dos proposiciones es verdera, sólo cuando ambas son verdaderas, en otro caso es falsa.	A B | A ∧ B 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	
La disyunción de dos proposiciones es falsa sólo en el caso que ambas son falsas, en caso contrario es verdadera.	A B | A ∨ B 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	
(Las anteriores son convenciones que se acostumbran en la matemática y también en el razonamiento usual).	(Las anteriores son las tablas de verdad para la negación, la conjunción y la disyunción. Se toman como punto de partida, es decir, son definiciones, para construir tablas a fórmulas proposicionales cualesquiera).	(Los artefactos [~], [∧] y [∨] se construyen para que sus estados posibles sean los representados en la figura. Al final de la tabla describimos una forma sencilla para construirlos).

INTERPRETACIONES USUALES	ALGUNOS SIMBOLOS USADOS DE LOGICA MATEMATICA	INTERPRETACION DE INTERES PARA NUESTRO TRABAJO
	¿Cómo se construye la tabla de-verdad de una fórmula proposi--cional cualquiera?. Veamos un - ejemplo.	¿Cómo se comporta el cir-cuito asociado a una fór-mula proposicional cual--quiera? Veamos un ejem--plo.
$a \leq 3$ $a = 2$ abreviatura de: $(a < 3$ ó $a = 3)$ y (no $a = 2$) Cuando a es 2 la proposi-ción es: $(2 < 3$ ó $2 = 3)$ y no $(2 = 2)$ ¿Cómo sabemos si es verda-dera o falsa?. Así:	E J E M P L O Consideremos la fórmula siguien te (que aparece escrita primero en la forma usual y después en-la forma de árbol). $(p \vee q) \wedge (\sim r)$  ¿Cómo obtenemos el valor de ver dad de la fórmula cuando, por - ejemplo, el de p es 1, el de q-es 0 y el de r es 1?	E J E M P L O A S O C I A D O El siguiente es el circui to asociado a la fórmula-de la columna central. -- (Compárese con la fórmula escrita en forma de árbol) .  ¿Cuál es el estado del -- alambre de salida cuando, por ejemplo, el del alam-bre p es 1, el de q es 0-

INTERPRETACIONES USUALES	ALGUNOS SIMBOLOS USADOS DE LOGICA MATEMATICA	INTERPRETACION DE INTERES PARA NUESTRO TRABAJO

(2 < 3 o 2 = 3) y (no 2 = 2)

V F V

VERDADERA FALSA

FALSA

vemos que la proposi-- ción 2 < 3 es verdadera, 2 = 3 es falsa y 2 = 2 es verdadera, y guia-- dos por la estructura- de la proposición compuesta, analizamos las proposiciones intermedias hasta que, al final, nos damos cuenta- que la proposición es- falsa.

(p ∨ q) ∧ (~ r)

1 0 1

1 0

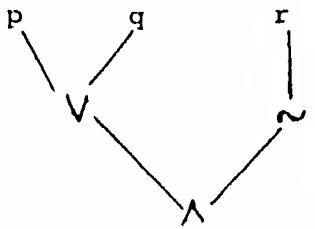
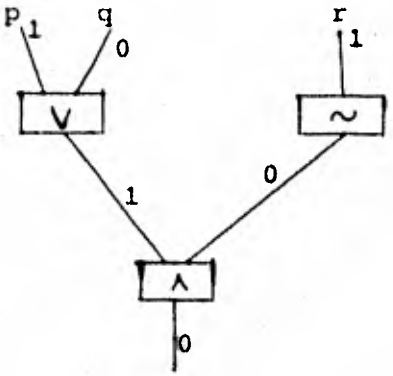
0

asignamos los valores de ver- dad a las letras proposicio- nales y, guiados por la es-- tructura de la fórmula, va- mos obteniendo valores de -- fórmulas intermedias y por - último el que nos interesa - (0 en nuestro caso). Se - - acostumbra decir que el pro- cedimiento de evaluación an- terior es "de adentro hacia- afuera", queriendo dar a en- tender que se inicia en las- partes más internas de la -- fórmula (las letras) y se -- termina en la parte externa. En la forma de árbol el pro-

y el de r es 1?.

Para averiguarlo basta que a partir de los alambres - de entrada, imitemos el -- flujo seguido por la co- - rriente y obtengamos así - los estados de los alam- - bres intermedios y por úl- timo el que nos interesa - (0 en nuestro caso).

Nótese que sobre el diagra- ma el proceso anterior es- "de arriba-hacia abajo" y- completamente análogo al -- proceso de evaluación de - la fórmula proposicional - asociada cuando está escri- ta en forma de árbol.

INTERPRETACIONES USUALES	ALGUNOS SIMBOLOS USADOS DE LOGICA MATEMATICA	INTERPRETACION DE INTERES PARA NUESTRO TRABAJO.
La situación anterior corresponde al sexto renglón de la Tabla de verdad que aparece en la columna central.	ceso de evaluación viene a ser entonces "de arriba hacia abajo".  Ya hemos construído el sexto-renglón de la Tabla de verdad, los otros se obtienen análogamente.	 (p ∨ q) ∧ (∼ r)

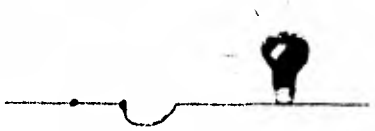
INTERPRETACIONES USUALES	ALGUNOS SIMBOLOS USADOS DE LOGICA MATEMATICA	INTERPRETACION DE INTERES PARA NUESTRO TRABAJO.

p	q	r	$(p \vee q) \wedge (\sim r)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Nótese que tenemos aquí una forma física para construir la Tabla de verdad de una fórmula proposicional. El estado representado es el correspondiente al sexto renglón de la Tabla de verdad. El circuito sólo presentará los ocho estados representados en la Tabla.

Vamos a ver ahora una forma sencilla para construir los artefactos [~], [∧] y [V].

Es casi seguro que quien esté leyendo esto tenga al alcance de la vista un interruptor para corriente eléctrica y más seguro es que cuando menos una vez lo haya accionado para "encender" o "apagar" un foco de luz. Lo que aquí nos interesa de tales aparatos es su aptitud para presentar cualquiera de los dos estados estables representados a continuación:



Estado 1. El interruptor permite el paso de corriente eléctrica (el foquito es auxiliar en el dibujo).

Estado 0. El interruptor impide el paso de corriente eléctrica.

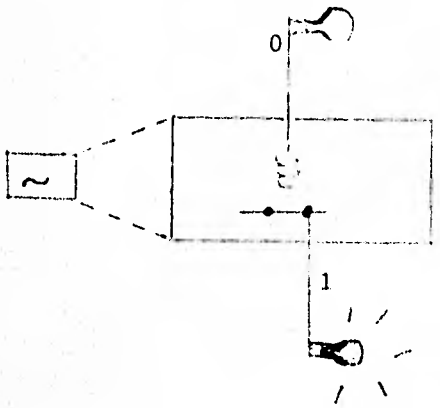
Otro tipo de aparato muy simple aunque no tan conocido es el electroimán. Los más sencillos se construyen enrollando alambre conductor (y si ésto se hace en un carrete de hierro se obtienen ciertas ventajas). También presentan dos estados estables:

Estado 1. Circula corriente por el alambre.

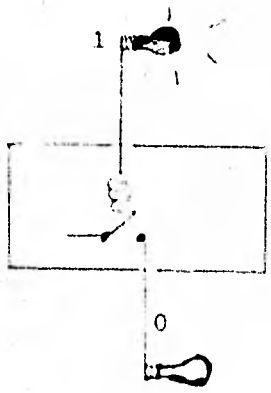
Estado 0. No circula corriente por el alambre.

Es bien interesante notar que los estados anteriores se distinguen no sólo porque circule o no corriente, o porque el foquito se encienda o no, sino porque en el estado 1 se genera un campo magnético. Este campo magnético se puede aprovechar para accionar un interruptor (construido con los materiales y en la forma adecuados).

Por ejemplo, veamos cómo se construye un aparato que se comporte como el que representamos con [~] ; los dos estados estables que presenta son:

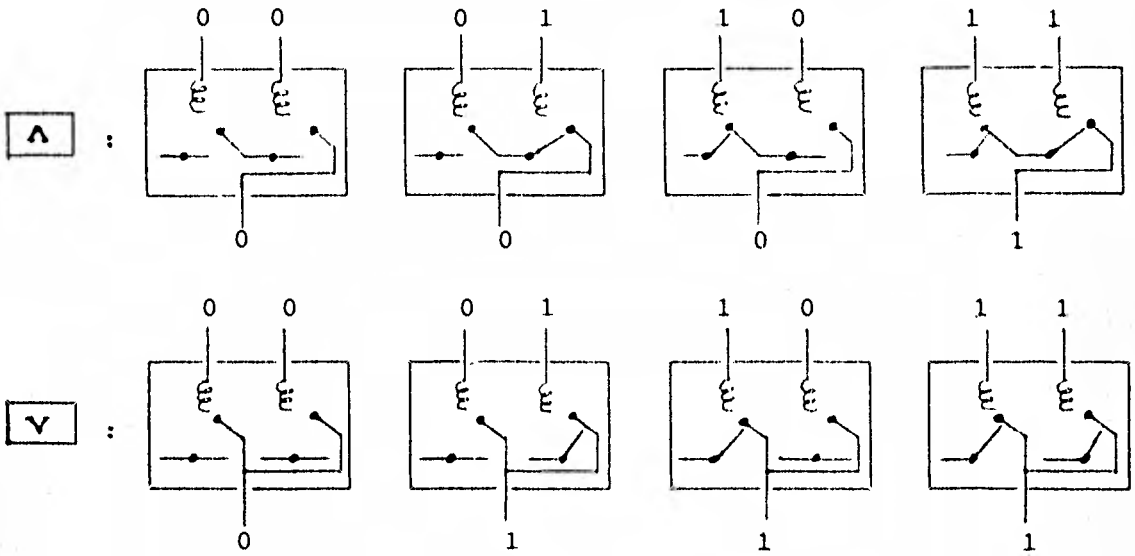


Cuando no fluye corriente en el alambre de entrada, el electroimán no genera campo magnético y el interruptor deja pasar la corriente (que se alimenta por otro lado a la línea del interruptor.



Cuando fluye corriente en el alambre de entrada, se genera un campo magnético que mantiene al interruptor en el estado que impide el paso de corriente al alambre de salida.

Y usando dos electroimanes y dos interruptores se pueden construir aparatos [∧] y [Y] ; veamos los cuatro estados estables de cada uno de ellos:



Esta es una forma sencilla bastante "palpable" de poder construir y ver "realizados" nuestros operadores.

Vayamos ahora adelante con el objetivo de esta sección. Las notaciones numéricas posicionales -entre las que están la binaria, la octal y la decimal- tienen ciertas propiedades comunes que resaltaremos con el siguiente ejemplo.

Supongamos que en un auditorio hay tantas mujeres como días tiene el año (no bisiesto) y tantos hombres como kilómetros (por carretera) separan a Guadalajara de Monterrey, ¿cuántas personas se encuentran reunidas?.

La manera usual de resolver este problema es como la siguiente:

.) Se representa el número de mujeres en notación decimal:

365

..) En la misma notación se representa el número de hombres:

758

...) Se recurre a un método para obtener la representación decimal del número de personas reunidas, a partir de las representaciones anteriores. (En otras palabras: aplicamos un algoritmo para sumar):

```
  1 1 1  ←—— (acarreo)
    3 6 5
+   7 5 8
-------
  1 1 2 3  ← (representación decimal del número de personas reunidas).
```

Pero la anterior no es la única manera de resolverlo. Es notorio que usando cualquiera otra notación posicional se puede proceder en forma muy parecida. Veamos cómo se hace usando el sistema binario.

I) Se representa el número de mujeres en notación binaria:

1 0 1 1 0 1 1 0 1

II) Se representa luego el número de hombres:

1 0 1 1 1 1 0 1 1 0

III) Se recurre a un método para obtener la representación binaria del total de personas reunidas.

```
  1 1 1 1 1 1 1 1 0 0 ← (acarreo)
    1 0 1 1 0 1 1 0 1
+ 1 0 1 1 1 1 0 1 1 0
-----------------------
  1 0 0 0 1 1 0 0 0 1 1 ←—— (representación binaria del número de personas reunidas.
```

Como se puede observar, la forma de sumar construyendo el resultado de derecha a izquierda, llevando el acarreo, no es exclusiva del sistema decimal, es una propiedad de las notaciones posicionales. Para facilitar el proceso de sumar podemos tener al lado una tabla de sumar en la notación que se esté usando. Por ejemplo, una tabla para sumar en sistema binario es la siguiente:

b	c	b + c	
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Conviene ampliarla a una con tres sumandos, pues de hecho siempre manejamos tres cantidades al sumar: el primer sumando, el segundo y el acarreo. En vista de la asociatividad de la suma podemos escribir sin ambiguedad que:

a	b	c	a + b + c	
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Esta tabla cubre todos los casos que se van presentando en cada paso del método de sumar que estamos usando.

NOTA IMPORTANTE: Notemos el parecido de las tablas de sumar en binario con las tablas de verdad en que los símbolos 0 y 1 denotan falso y verdadero, respectivamente.

¿Se puede aprovechar este parecido de alguna manera?.

Sí: a continuación veremos cómo el doble uso de los símbolos mencionados permite una combinación de la aritmética y la Lógica Matemática aplicable en el diseño de un circuito sumador, similar a los usados por las computadoras electrónicas.

Empecemos partiendo en dos nuestra última tabla de sumar -a - la cual ya le habíamos notado cierto parecido con las tablas de verdad- en la forma ilustrada a continuación:

a	b	c	a + b + c
0	0	0	0 0
0	0	1	0 1
0	1	0	0 1
0	1	1	1 0
1	0	0	0 1
1	0	1	1 0
1	1	0	1 0
1	1	1	1 1

"acarreo" de

a	b	c	a + b + c
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

6.1) Tabla para determinar el acarreo de a+b+c

a	b	c	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

6.2) Tabla para ir consyendo el resultado de una suma.

Estas dos últimas tablas de sumar y "acarreo" sí que se parecen a las Tablas de verdad: no lo son porque aquí los símbolos 0 y 1 denotan números, que no son valores de verdad. Además a, b y c no son letras proposicionales. Si queremos verlas como Tablas de verdad, tan sólo cambiemos la interpretación a los símbolos 0 y 1 que aparecen en ellas. Pensémoslos entonces como valores de verdad de letras proposicionales p, q y r de fórmulas proposicionales construidas en términos de dichas letras:

p	q	r	?
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

6.3) Corresp. a la 6.1

p	q	r	?
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

6.4) Corresp. a la 6.2

Pero... ¿de qué fórmulas proposicionales?.

Sabemos construir la Tabla de verdad cuando nos dan una fórmula, ahora se nos presenta el problema inverso: conocemos la Tabla y queremos una fórmula.

Por fortuna la Lógica Matemática resuelve el problema pues desarrolla métodos para construir fórmulas con la propiedad deseada; ilustraremos uno de tales métodos generando una fórmula a la Tabla 6.3. (ver el cap. III, parte a), secc. 6).

La idea es reducir el problema al de construir fórmulas a tablas más simples, como lo son aquéllas con un sólo 1. En el-

ejemplo nos servirán las que corresponden a las columnas (c1), (c2), (c3) y (c4). (hay una tabla por cada 1 de la tabla original).

p	q	r	?	?	?	?	?
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	1	1	1	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	1	0	0	1	0
1	1	1	1	0	0	0	1
				(c1)	(c2)	(c3)	(c4)

A estas tablas ya es más fácil construirles fórmulas, fijándonos en la tabla del operador "$\wedge$", cuya propiedad es que sólo vale 1 cuando todos los valores también lo son.

p	q	r	?	$(\sim p)\wedge q\wedge r$	$p\wedge(\sim q)\wedge r$	$p\wedge q\wedge\sim r$	$p\wedge q\wedge r$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	1	1	1	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	1	0	0	1	0
1	1	1	1	0	0	0	1

La disyunción de las fórmulas obtenidas,

(6.5)... $(\sim p\wedge q\wedge r)\vee(p\wedge\sim q\wedge r)\vee(p\wedge q\wedge\sim r)\vee(p\wedge q\wedge r)$

es una fórmula que tiene la tabla dada. (ver el cap. III, parte a), secc. 6).

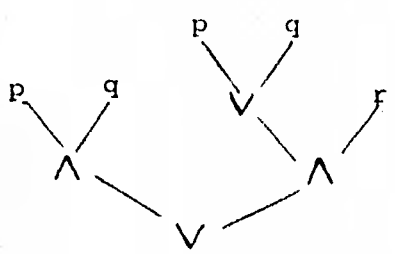
Usando el mismo método se construye la siguiente fórmula para la tabla 6.4:

(6.6)... $(\sim p \wedge \sim q \wedge r) \vee (\sim p \wedge q \wedge \sim r) \vee (p \wedge \sim q \wedge \sim r) \vee (p \wedge q \wedge r)$.

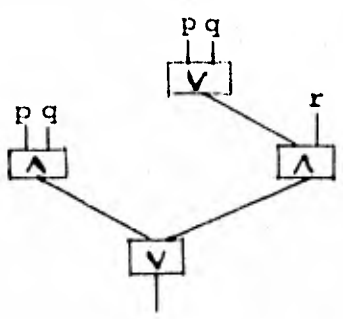
Hagamos un recuento de lo que hemos hecho. La tabla de sumar original la hemos partido en dos nuevas tablas (la de "acarreo" y la de "resultado"), las cuales hemos reinterpretado como Tablas de verdad y les hemos construido fórmulas.

Cabe preguntar, ¿para qué se está haciendo esto?. La respuesta es simple: ésto da un camino para construir circuitos que generen físicamente las tablas 6.3 y 6.4 y, por lo tanto, las Tablas 6.1 y 6.2; juntando luego dichos circuitos se tendrá uno que genere la tabla de sumar. Sigamos ese camino.

Primero construyamos un circuito para la fórmula 6.5. Para ello conviene escribir la fórmula en forma de árbol, con lo que inmediatamente se tiene la forma del circuito; pero antes es útil pasar la fórmula a otro equivalente (que tenga la misma Tabla de verdad) más simple (ver el cap. III, parte a), secc. 6). Por ejemplo, la fórmula $(p \wedge q) \vee ((p \vee q) \wedge r)$... 6.7. Tiene la misma tabla de verdad que la 6.5 y es mucho más simple. Construimos su circuito,

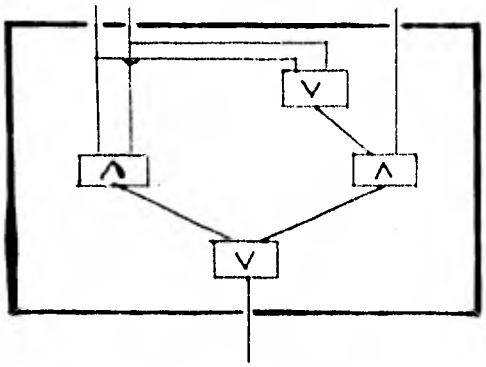


Forma de Arbol de 6.7

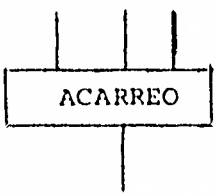


Circuito para 6.7.

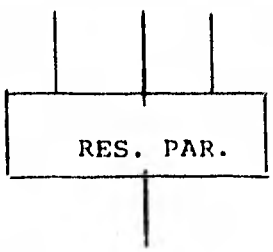
Para no tener demasiados alambres de entrada, interconectamos aquellos que presenten estados iguales, es decir, aquellos -- que corresponden a la misma letra proposicional:



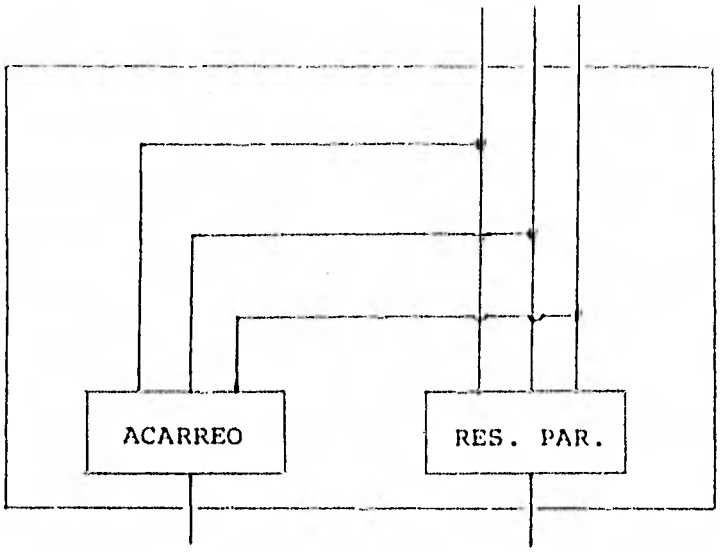
Llamaremos ACARREO a cualquier circuito como el anterior (o - cualquier otro que se comporte igual que él) y lo representaremos así:



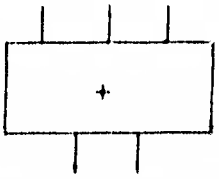
Similarmente se puede construir un circuito para la fórmula - 6.6 (o cualquiera otra que tenga la misma Tabla de verdad). - Para recordar que un circuito tal se comporta como lo describe la tabla 6.2 lo llamaremos RESULTADO PARCIAL y lo representaremos así:



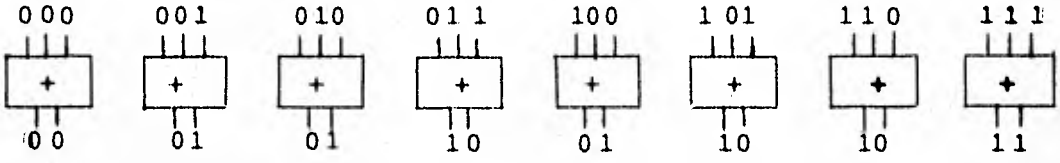
Ahora juntamos circuitos de los anteriores en la forma indicada a continuación:



formando de esta manera artefactos que representaremos así:



y que se comportan tal como lo describe la tabla de sumar que obtuvimos en un principio. Es decir, sus ocho estados posibles son los representados a continuación:

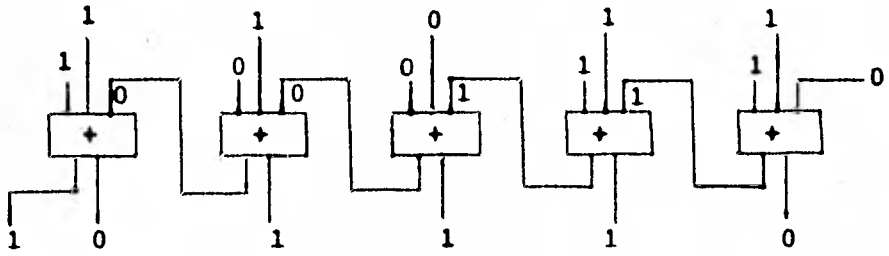


Para terminar veamos cómo se interconectan circuitos del tipo anterior para construir el anunciado circuito sumador. Antes de hacerlo conviene mencionar que hay maneras de almacenar representaciones binarias de números y de que esa información, automáticamente, se alimente a los alambres de entrada con la consiguiente determinación de los estados de los alambres de salida.

La interconexión la mostramos por el siguiente diagrama que ilustra, por ejemplo, el estado del circuito asociado a la suma 27 + 19 = 46, pero en binario; es decir, de la suma:

```
  1   0   0   1   1   0   <-- (acarreo)
      1   1   0   1   1
      1   0   0   1   1
  ---------------------
  1   0   1   1   1   0   <-- (resultado)
```

Se recomienda seguir el diagrama a la manera de la suma anterior: de derecha a izquierda.



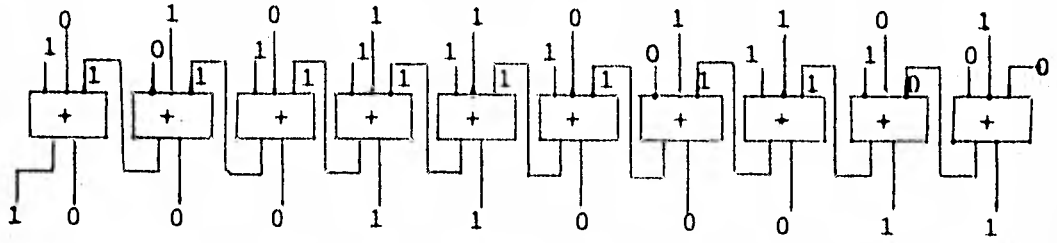
Fijémonos que el artefacto  que hemos construido, con sus ocho estados posibles, es el que nos resuelve en realidad la suma. Resolvamos la suma que tuvimos como ejemplo en un principio. En decimal es:

```
   365   mujeres
+  758   hombres
  ----
  1123   personas
```

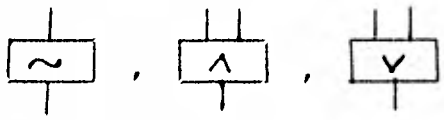
En binario tenemos:

1	1	1	1	1	1	1	1	0	0	←——(acarreo)
	0	1	0	1	1	0	1	1	0	1	
	1	0	1	1	1	1	0	1	1	0	
1	0	0	0	1	1	0	0	0	1	1	←——(resultado

Entonces el estado del circuito asociado a esta suma está representado por el siguiente diagrama:



Debemos confesar que la gran mayoría de los artefactos



y similares, usados en la construcción de las computadoras electrónicas modernas, no se fabrican en la forma descrita aquí, debido a que serían muy ineficientes tanto en rapidez para cambiar de estado como en ahorro de energía y espacio. Mas no por esto se deben despreciar, pues son usadísimos en otros terrenos apropiados a ellos, por ejemplo en la construcción de circuitos lógicos para controlar multitud de procesos industriales. Por medio de este modelo físico, más sencillo en su representación sobre el papel, hemos tratado de dar idea en esta sección de las formas en que se construyen di-

chos artefactos en la realidad combinando transistores y resistencias en vez de interruptores y electroimanes.

Creo que ejemplos como el aquí presentado ayudan a quitar el velo de misterio que rodea a las computadoras y proporcionan atractivas puertas de entrada al estudio de los sistemas de notación y la lógica matemática, tan importantes en la comprensión de la matemática.

CAPITULO III

LOGICA MATEMATICA

a) El cálculo proposicional.

b) Vectores binarios.

COMENTARIO AL CAPITULO III.

Desarrollaré en este capítulo tan sólo la parte de la Lógica-Matemática que utilizaré posteriormente, el cálculo proposi-- cional. Otros temas más desarrollados de la Lógica, tal como el cálculo de predicados de primer orden, tienen también íntima conexión con el tema de las computadoras, sobre todo en la parte de "Prueba Lógica"; sin embargo, lo dejo fuera del contexto de este trabajo a fin de concentrar el análisis en los-métodos de cálculo con proposiciones o sentencias preparando-un "puente" para el encuentro con los algoritmos.

Así pues, seré breve y conciso en la exposición de los conceptos básicos del cálculo proposicional, introduciéndolo más como un arma futura que como tema de discusión. La exposición-será no-axiomática, reservando la sección a)8. para un enfoque axiomático. En la parte b) se examina una estructura muy particular, la de vectores binarios; aunque no es parte de la lógica matemática, la estudio en este capítulo desde el punto de vista de conjuntos, números y del cálculo proposicional. - Dicha estructura nos introduce al lenguaje binario de las computadoras.

a) EL CALCULO PROPOSICIONAL.

1. Conceptos básicos.
2. Fórmulas bien formadas.
3. Tablas de verdad.
4. Argumentación y evaluación.
5. Equivalencia lógica y consecuencia lógica.
6. Formas normales.
7. Notación polaca y el árbol de una fórmula.
8. Conjuntos minimales de conectivos.
9. Un enfoque axiomático.

1. CONCEPTOS BASICOS.

Nos ocuparemos aquí de proposiciones declarativas que serán - "ciertas" (denotado "V") o "falsas" (denotado "F"), y con - métodos de combinarlas para deducir de ellas nuevas proposi-- ciones.

Definición 1.1

Sea p una proposición. La negación de p, denotada $\sim p$, es otra proposición, definida por la siguiente tabla:

p	$\sim p$
V	F
F	V

Tabla 1.1 negación.

Definición 1.2

Sean p y q proposiciones. La conjunción de p y q, denotada $p \wedge q$, es otra proposición, definida por la tabla:

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

Tabla 1.2 conjunción.

Definición 1.3

Sean p y q proposiciones. La disyunción inclusiva o disyunción de p y q, denotada $p \vee q$, es otra proposición, definida por la tabla:

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

Tabla 1.3 disyunción.

Definición 1.4.

Sean p y q proposiciones. La disyunción exclusiva o inequivalencia de p y q, denotada $p \nleftrightarrow q$, es otra proposición, definida según la tabla:

p	q	$p \nleftrightarrow q$
V	V	F
V	F	V
F	V	V
F	F	F

Tabla 1.4 inequivalencia.

Las cuatro operaciones que hemos definido nos proporcionan el enlace necesario con las álgebras booleanas del capítulo anterior. Negación, conjunción, disyunción e inequivalencia se corresponden, respectivamente con las operaciones booleanas de

complementación, multiplicación, y las dos adiciones + y #.

cálculo prop.		álgebra booleana.
$\sim$	$\longmapsto$	$'$
$\wedge$	$\longmapsto$	$\bullet$
$\vee$	$\longmapsto$	$+$
$\nmid$	$\longmapsto$	$\#$

Si pensamos en p y q no como representando oraciones, sino como variables que toman los valores V y F, entonces $\wedge$, por ejemplo, es un mapeo de $\mathcal{T} \times \mathcal{T}$ en $\mathcal{T}$, donde $\mathcal{T} = \{ V, F \}$.

Hay cuatro elementos en $\mathcal{T} \times \mathcal{T}$, a saber: $\langle V, V \rangle$, $\langle V, F \rangle$, $\langle F, V \rangle$ y $\langle F, F \rangle$. Cualquier mapeo de $\mathcal{T} \times \mathcal{T}$ en $\mathcal{T}$ debe asignar un valor - o V o F - a cada uno de estos pares ordenados. Como cada uno de estos cuatro elementos puede tomar uno de los dos valores independientemente, hay 2 x 2 x 2 x 2 = 16 posibles mapeos de $\mathcal{T} \times \mathcal{T}$ en $\mathcal{T}$. Los mapeos que hemos dado y llamado disyunción, conjunción e inequivalencia, son solamente tres de éstos. En la tabla 1.5 se enumeran todos los mapeos posibles.

		T	$\vee$	$\subset$	p	$\supset$	q	$\sim(\#)$	$\wedge$	$\sim(\wedge)$	$\nmid$	$\sim$q	$\not\supset$	$\sim$p	$\not\subset$	$\sim(\vee)$	∇
p	q	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
V	V	V	V	V	V	V	V	V	V	F	F	F	F	F	F	F	F
V	F	V	V	V	V	F	F	F	F	V	V	V	V	F	F	F	F
F	V	V	V	F	F	V	V	F	F	V	V	F	F	V	V	F	F
F	F	V	F	V	F	V	F	V	F	V	F	V	F	V	F	V	F

Tabla 1.5 Mapeos de $\mathcal{T} \times \mathcal{T} \longmapsto \mathcal{T}$

Examinando esta tabla encontramos las operaciones o conectivos que ya definimos:

- la disyunción $\vee$ está en la columna 2.
- la conjunción $\wedge$ está en la columna 8.
- la inequivalencia está en la columna 10.

También encontramos que:

- p está representado en la columna 4.
- $\sim p$ en la columna 13.
- q en la columna 6.
- $\sim q$ en la columna 11.
- la columna 1 es "universalmente cierto" (tautología).
- la columna 16 es "universalmente falso".(contradicción).
- el conectivo "$\supset$" en la 5, que luego definiremos, y- su recíproco "$\subset$" en la 3.

Una observación importante en la tabla 1.5 es que:

> La columna n es la negación de la columna 17-n, $1 \leq n \leq 16$; en el mismo sentido que $\sim p$ es la- negación de p.

así,

- como la columna 2 representa la disyunción $p \vee q$, la - columna 17-2 = 15 representa la antidisyunción o ni -- (no o): "no es cierto que p o q o ambos". Se denota - por la flecha de Pierce: $p \downarrow q$. (ni p ni q) (negación conjunta).

- Análogamente, la columna 9 = 17 - 8 representa la negación de la conjunción y se conoce como la anticonjun-- ción o nand (de: not and). Se denota por el trazo de -- Sheffer: $p|q$. (no p o no q) (negación alterna) o bien, p incompatible con q.

- La negación de la columna 10 = 17 - 7 (inequivalencia) es la bicondicional o equivalencia. Se denota por -- $p \equiv q$, (p si y sólo si q).

Definición 1.5

Sean p y q proposiciones. La condicional si p entonces-q o implicación (material) es la proposición definida -- por la columna 5 de la tabla 1.5. A p se le llama antecedente y a q el consecuente de la condicional. Se denota $p \supset q$.

La operación en la columna 3 es la misma con los papeles de p y q intercambiados. Se le suele llamar implicación recíproca y se escribe $p \subset q$.

2. FORMULAS BIEN FORMADAS.

En la tabla 1.5 de la sección anterior, dos cosas se hacen - evidentes.

La primera es que sus 16 columnas representan todas las asignaciones posibles de valores de verdad dependientes de tan sólo dos argumentos p y q, luego no importa cuán compleja pueda ser una proposición construida, partiendo de dos proposiciones básicas, siempre será en cierto sentido equivalente a una de éstas 16.

La otra es que podríamos construir tablas análogas para 3 o - más variables. Pero tales tablas aumentan el número de columnas considerablemente. Por ejemplo, para tres variables habría $2^{2^3} = 256$ columnas y para cuatro habría $2^{2^4} = 65,536$ columnas.

Afortunadamente no hay necesidad alguna de construir tales tablas; en la sección 6, veremos que todas esas funciones --

pueden expresarse en términos de las que hemos construido. - Para tal efecto, comenzaremos por definir qué son las fórmu-- las bien formadas (f.b.f.), éstas son, intuitivamente, cade-- nas de símbolos que tienen gramaticalmente sentido, según - - nuestra "gramática" que construiremos. El sistema de puntuación básico que usaremos consiste en paréntesis.

Estamos desarrollando un lenguaje netamente simbólico. Necesitaremos hacer proposiciones dentro de este lenguaje y tam-- bién proposiciones acerca del lenguaje; tendremos teoremas - dentro del lenguaje y teoremas acerca del lenguaje, trabajare mos, pues, en dos niveles distintos de discurso, el llamado - "lenguaje" o "teoría" y el "metalenguaje" o "metateoría". Usa remos el Metalenguaje para definir en forma precisa los modos de expresión dentro del lenguaje; pero no introduciré un meta metalenguaje para discutir el metalenguaje. Las nociones de- proposición y prueba dentro del metalenguaje las daré en una- forma menos precisa y bastante intuitiva.

En <u>la teoría</u>, tenemos los siguientes símbolos:

conectivos lógicos: $\sim, \wedge, \vee, \nleftrightarrow, \supset, \downarrow, |, \equiv, \subset, \not\supset, \not\subset$

paréntesis: (,)

variables proposicionales: $p, q, r, p_1, q_1, r_1, \ldots$

En el <u>metalenguaje</u>, para denotar fórmulas bien formadas usare mos las letras A, B, C, $A_1, \ldots$

Por ejemplo, "$(A \supset B)$" no es una expresión en la teoría, si- no en la metateoría que denota cualquier miembro deseado, den tro de la clase de fórmulas de la teoría, que se obtienen al- poner expresiones de la teoría en el lugar de las letras A y- B.

Definición 2.1

Una fórmula bien formada (f.b.f.) del cálculo proposicional, se define por la siguiente recursión:

1. Una variable proposicional aislada es una f.b.f.
2. Si A es una f.b.f., entonces $(\sim A)$ es una f.b.f.
3. Si A y B son f.b.f., entonces las siguientes son f.b.f.:

   $(A \wedge B)$, $(A \vee B)$, $(A \supset B)$, $(A \equiv B)$, $(A \subset B)$,
   $(A \not\supset B)$, $(A \not\subset B)$, $(A \not\equiv B)$, $(A \mid B)$ y $(A \downarrow B)$.

4. Una cadena de símbolos es una f.b.f., si y sólo si el serlo se sigue de un número finito de aplicaciones de las reglas 1, 2 y 3.

Como las fórmulas que se trabajan en este tema son, ciertamente, para "consumo humano", es conveniente entonces hacer la fórmula lo más legible posible. Así, tenemos varias convenciones con respecto a la notación:

- Convenimos en que si A es una f.b.f. que no sea variable proposicional, suprimimos los paréntesis externos.
- Convenio de jerarquía u orden de precedencia.

  Para captarlo mejor, veamos un ejemplo:

  En aritmética la expresión 3 x 4 + 5 es ambigua, pero tendemos a interpretarla como (3 x 4) + 5 y no como -- 3 x (4 + 5), dando así precedencia a la multiplicación respecto a la adición.

  En Cálculo Proposicional, convenimos la precedencia de los conectivos como sigue: $\sim, \wedge, \vee, \supset, \equiv$ en el orden usual, (los otros conectivos no los incluyo pues casi son inusuales).

Así, $\sim p \supset q$ significa $(\sim p) \supset q$, y no $\sim (p \supset q)$.

- Convenimos en que para dos ocurrencias (o más) consecutivas de un mismo conectivo, el de más a la izquierda toma precedencia sobre el más a la derecha, en ese orden:

  así, $p \supset q \supset r \supset s$ es $((p \supset q) \supset r) \supset s$.

3. TABLAS DE VERDAD.

Una tabla de verdad es un arreglo tabular que muestra el valor de verdad de una f.b.f. para cada asignación de valores de verdad a sus proposiciones componentes.

Por ejemplo, la tabla 3.1 es la tabla de verdad para la f.b.f. $(p \supset (p \supset p)) \supset p$.

p	$(p \supset (p \supset p)) \supset p$		
V	V	V	V
F	V	V	F

Tabla 3.1

Definición 3.1

- Si para toda asignación de valores a sus variables una f.b.f. A tiene el valor V, entonces se llama una <u>tautología</u>.
- Se denota por $\models A$.
- Si siempre tiene el valor F, entonces se llama una <u>contradicción</u>.
- Una f.b.f. que no es tautología ni contradicción se llama <u>contingente</u>.

Por ejemplo, la tabla 3.2 nos muestra que la f.b.f.- $(p \wedge q) \supset (p \vee \sim r)$ es una tautología.

p	q	r	$p \wedge q$	$\supset$	$p \vee \sim r$
V	V	V	V	V	V
V	V	F	F	V	V
V	F	V	F	V	V
V	F	F	F	V	V
F	V	V	F	V	F
F	V	F	F	V	V
F	F	V	F	V	F
F	F	F	F	V	V

Tabla 3.2

Nótese que como la columna final o principal de la tabla de - valores de una tautología consiste solamente en V, y la de -- una contradicción, solamente en F, entonces resulta claro que:

Una f.b.f. A es una tautología

si y sólo si

su negación $\sim$ A es una contradicción.

4. ARGUMENTACION Y EVALUACION.

El problema básico, en el cálculo de las funciones de verdad- como arte aplicado, es el de representar y evaluar las proposiciones y argumentos de la vida cotidiana.

Consideremos, por ejemplo, la proposición siguiente:

Iremos a la reunión, si y sólo si hay fondos disponibles y si la reunión no interfiere con nuestro trabajo o hay -

una sesión en la reunión que está directamente relacionada con nuestro trabajo.

Un análisis lógico adecuado de esta proposición puede comenzar con el simbolismo:

$$W \equiv F \wedge (\sim C \vee S)$$

donde las letras denotan las siguientes proposiciones:

W : iremos a la reunión.
F : hay fondos disponibles.
C : la reunión interfiere con el trabajo.
S : hay una sesión relacionada.

La cuestión básica que nos preguntamos es bajo qué condiciones es cierta esta proposición. El valor de verdad de esta expresión depende claramente de los de W, F, C y S; la expresión representa una función de verdad que va de $\mathcal{T} \times \mathcal{T} \times \mathcal{T} \times \mathcal{T}$ en $\mathcal{T}$. Tenemos cuatro variables proposicionales y a cada una podemos asignarle el valor V o F, independientemente, hay entonces $2^4 = 16$ filas en la tabla de verdad de la función.

Analizando dicha tabla 4.1 vemos que la proposición es cierta solamente bajo las siguientes condiciones:

i) W y F son ambas ciertas, y o C es falsa o S es cierta.

ii) Tanto W como F son falsas.

iii) W y S son falsas, F y C son ciertas.

De esta forma sabemos cuando la proposición se cumple.

W	F	C	S	F	$\wedge$	$(\sim C \vee S)$	$\equiv$
V	V	V	V		V	V	V
V	V	V	F		F	F	F
V	V	F	V		V	V	V
V	V	F	F		V	V	V
V	F	V	V		F	V	F
V	F	V	F		F	F	F
V	F	F	V		F	V	F
V	F	F	F		F	V	F
F	V	V	V		V	V	F
F	V	V	F		F	F	V
F	V	F	V		V	V	F
F	V	F	F		V	V	F
F	F	V	V		F	V	V
F	F	V	F		F	F	V
F	F	F	V		F	V	V
F	F	F	F		F	V	V

Tabla 4.1

5. EQUIVALENCIA LOGICA Y CONSECUENCIA LOGICA.

Definición 5.1

Dos proposiciones A y B son <u>Lógicamente Equivalentes</u> (A eq B), si y sólo se tienen la misma Tabla de verdad.

Notemos que esto no implica que A y B "signifiquen" la misma-cosa. Por ejemplo, las proposiciones:

"Si x es un entero entonces 2x es un entero".

"Si x es primo distinto de 2, entonces x es impar."

ambas tienen la misma forma lógica $(p \supset q)$ y son lógicamente-equivalentes.

En la definición dada va implícita la idea de que las variables que aparecen en A son las mismas que las que aparecen en B, pues si los valores de A dependen de los valores, digamos, de r, pero B no contiene a r como variable, entonces, generalmente A y B no serán lógicamente equivalentes.

Teorema 5.1

A eq B si y sólo si $\models A \equiv B$

Prueba.

Supongamos que $\models A \equiv B$.

Independientemente de la asignación de valores de verdad a las variables en $A \equiv B$, esta fórmula tiene el valor V.- Entonces, para cada asignación particular de valores de-verdad a estas variables, o A tiene el valor V y por tanto B también, o ambas tienen el valor F. Luego A y B -- tienen la misma tabla de verdad y, por ende, A eq B.

Supongamos que A eq B.

Consideremos cualquier asignación de valores de verdad a las variables en A y B. Si para tal asignación A tiene-el valor V, entonces B también tiene el valor V y, por - tanto, $A \equiv B$ tiene el valor V. Si A tiene el valor F, como A eq B, entonces B tiene el valor F y, por tanto, -- $A \equiv B$ tiene el valor V. Por tanto $A \equiv B$ tiene siempre el-valor V $\therefore \models A \equiv B$.

Q.E.D.

Ya vimos el concepto de "equivalencia lógica": tener la misma tabla de verdad. ($\models A \equiv B$). Veamos ahora el concepto de "consecuencia lógica".

Definición 5.2

B es una consecuencia lógica de A ($A \models B$) si para toda asignación de valores de verdad a las variables de A (y B) tal que A toma el valor V, entonces B también toma el valor V.

Intuitivamente, lo que nos dice esto es que: "siempre que A sea cierto, entonces B debe ser necesariamente cierto.

Teorema 5.2

$A \models B$ si y sólo si $\models A \supset B$

La prueba es análoga a la del Teorema 5.1.

La generalización para la definición 5.2 la tenemos de la siguiente manera:

Definición 5.2 (Generalizada).

$A_1, \ldots, A_n \models B$, si y sólo si siempre que

$A_1, \ldots, A_n$ son todas ciertas, entonces B también es cierta.

Para el teorema 5.2 tenemos la siguiente generalización:

Teorema 5.2 (Generalizado).

$A_1, \ldots, A_n \models B$ si y sólo si $\models A_1 \supset (\ldots \supset (A_n \supset B) \ldots)$

Lema

$A_1 \supset (\ldots (A_n \supset B) \ldots)$ eq $A_1 \wedge \ldots \wedge A_n \supset B$

Corolario 1

$A_1, \ldots, A_n \models B$ si y sólo si $\models A_1 \wedge \ldots \wedge A_n \supset B$

Corolario 2

Si $A, B \models C$ entonces $B, A \models C$

Corolario 2 (Generalizado)

Si $A_1, A_2, \ldots, A_n \models B$ entonces $A_{i_1}, A_{i_2}, \ldots, A_{in} \models B$

donde $i_1, i_2, \ldots i_n$ es una permutación cualquiera de los enteros 1, 2,..., n.

En estas partes generalizadas, las pruebas las omito intencionadamente para pasar a ver las consecuencias de la equivalencia y consecuencia lógicas mediante dos ejemplos. En primer lugar, nos permiten movernos, hasta cierta medida, entre las ideas de consecuencia lógica y tautología. En particular, se pueden establecer tautologías y usarlas para generar reglas de inferencia informales. Este método tiene la ventaja práctica de que gran parte de la comprobación de la tabla de verdad puede eliminarse.

Ejemplo 5.1

Desarrollemos la regla de inferencia conocida como modus ponens.

El primer paso es establecer $\models A \wedge (A \supset B) \supset B$ por la tabla 5.1

A	B	$A \wedge$	$(A \supset B)$	$\supset B$
V	V	V	V	V
V	F	F	F	V
F	V	F	V	V
F	F	F	V	V

Tabla 5.1

Usando luego el corolario 1, podemos asegurar que, A, $A \supset B \not\models B$. Es decir, en la hipótesis de las fórmulas A y $A \supset B$, podemos concluir B. En particular, si $\models A$ y $\vdash A \supset B$, entonces $\models B$. (Nótese que ésta es la primera fila de nuestra tabla de verdad).

Ejemplo 5.2

Para ilustrar el análisis de un argumento, examinemos el siguiente problema:

Los deseos son caballos con tal de que los caballos no puedan volar. Los mendigos no cabalgarán con tal de que los deseos no son caballos. Si no puede darse el caso de que a la vez los mendigos cabalguen y que los deseos no sean equinos, entonces los caballos pueden volar. Si la falta de capacidad de los caballos para el vuelo y el no cabalgar de los mendigos no pueden ponerse como alternativas válidas, entonces los mendigos no siempre son ricos. Pero los mendigos cabalgan. Pregunta: ¿Son los mendigos ricos?.

Usaré el siguiente simbolismo:

los deseos son caballos	D
los caballos pueden volar	E
los mendigos cabalgan	M
los mendigos siempre son ricos	R

El argumento se hace entonces como sigue:

A_1: $\sim E \supset D$
A_2: $\sim D \supset \sim M$
A_3: $\sim (M \wedge \sim D) \supset E$
A_4: $\sim (\sim E \vee \sim M) \supset \sim R$
A_5: M

B : R (?) (conclusión).

Se quiere que éste sea un argumento de la forma $A_1, \ldots A_5 \models B$.- Debemos pues examinar las condiciones bajo las cuales $A_1, \ldots A_5$ son ciertas. Usaré la notación $A \longmapsto \varphi$, donde φ va a ser- V o F, para indicar que a A se le da el valor de verdad V.

1.- $A_5 \longmapsto V$ si y sólo si $M \longmapsto V$

2.- Por 1, $A_2 \longmapsto V$ si y sólo si $D \longmapsto V$

3.- Por 1 y 2, $A_3 \longmapsto V$ si y sólo si $E \longmapsto V$

4.- Por 1 y 3, $A_4 \longmapsto V$ si y sólo si $\sim R \longmapsto V$

o sea: si y sólo si $R \longmapsto F$

5.- De 4 resulta que $A_1, \ldots, A_5 \models B$ si y sólo si $R \longmapsto F$ o sea $\sim R \longmapsto V$; por tanto la respuesta a la pregun ta final del problema es que: los mendigos no siem- pre son ricos.

Pasemos ahora al siguiente tema que trata sobre la representa ción de las funciones lógicas.

6. FORMAS NORMALES.

Veremos una técnica para representar funciones lógicas de más de dos variables usando solamente las funciones binarias.

Hasta el momento contamos ya con 10 conectivos binarios más - la negación; con tal cuenta, si A y B son fórmulas complejas- de muchas variables y diversos conectivos, la tarea de esta-- blecer si son lógicamente equivalentes o no puede ser tediosa y difícil de realizar en una computadora. Para simplificar - la tarea sería conveniente desarrollar una forma estándar de- representación para las funciones; teóricamente sería posible- transformar fácilmente A y B en formas estándar A' y B' tales que A eq B fuera de comprobación o rechazo rápido por examen- de A' y B'. Esto es posible, aunque lo de "fácilmente" no --

puede considerarse una palabra apropiada. Otra razón a favor de la forma estándar, es la de que dada una f.b.f. A, es conveniente seleccionar una f.b.f. B con BeqA, que sea en cierto sentido mínima.

Por ejemplo, $[(p \wedge q) \supset (\sim p \vee (p \mid \sim q))] \dashv [\sim p \supset ((q \subset p) \downarrow \sim q)]$ debe simplificarse a una de las 16 formas de la tabla 1.5 ya que envuelve solamente dos variables (de hecho, es equivalente a $p \downarrow q$).

Las formas estándar proporcionan un camino sistemático de encontrar las formas mínimas, las que veremos se relacionan con las formas normales introducidas en la parte de álgebras booleanas, (sección 3 del Cap. II b)). En realidad, como la lógica que desarrollamos constituye un conjunto de funciones booleanas, todas las técnicas discutidas en tales álgebras pueden usarse en el cálculo proposicional. Recíprocamente, la discusión que aquí hacemos puede aplicarse "mutatis mutandis" a las funciones booleanas, correspondiendo:

- $+$ la suma booleana a la disyunción $\vee$
- $\cdot$ el producto booleano a la conjunción $\wedge$

De las varias formas estándar (o canónicas o normales) consideraremos cuatro.

Suponemos que las variables proposicionales están dadas en algún orden, al que llamaremos orden alfabético, digamos p, q, r, p_1, q_1, r_1,...

Definición 6.1

▲) Una fórmula es una conjunción elemental, si es de la forma $A_1 \wedge A_2 \wedge \ldots \wedge A_n$, donde:

1. Toda A_i es o una variable proposicional o la negación de una variable proposicional;

2. Ninguna variable proposicional aparece en más de una A_j;

3. Aquellas variables proposicionales que aparecen están en orden alfabético, es decir, si $i < j$ entonces la variable proposicional que aparece en A_i precede a la que está en Aj en el alfabeto.

▲▲) Una fórmula está en forma normal disyuntiva (FND), si es de la forma $B_1 \vee B_2 \vee \ldots \vee B_m$, donde:

1. Cada Bi es una conjunción elemental.
2. No hay dos B_i que sean iguales.
3. Si $B_i = A_{i_1} \wedge A_{i_2} \wedge \ldots \wedge A_{i_n}$

y

$$B_j = A_{j1} \wedge A_{j2} \wedge \ldots \wedge A_{jn}$$

y

$$A_{i1} = A_{j1}$$

$$A_{i2} = A_{j2}$$

$$\vdots$$

$$A_{i_{k-1}} = A_{j_{k-1}}$$

y

$$A_{ik} \neq A_{jk}$$

Entonces:

$i < j$ si y sólo si se verifica alguna de las siguientes alternativas:

a) A_{ik} no existe (i. e. $k-1 = n_i$)

b) $A_{ik} = p$ y $A_{jk} = \sim p$ para alguna variable proposicional p.

c) La variable proposicional que aparece en A_{ik} precede a la que aparece en A_{jk}.

▲▲▲) Una f.b.f. está en <u>forma normal disyuntiva plena</u> (FNDP) si está en FND y toda variable proposicional que aparece en una cualquiera de las conjunciones elementales aparece en todas las conjunciones elementales.

Definición 6.2

Los términos.

▲) <u>disyunción elemental</u>

▲▲) <u>forma normal conjuntiva</u> (FNC)

y

▲▲▲) <u>forma normal conjuntiva plena</u> (FNCP)

Se definen análogamente a (6.1), con la disyunción y la conjunción intercambiadas.

Nótese que la expresión más simple para una tautología, $p \vee \sim p$, es una FND, pero no una FNC ya que p y $\sim$ p no pueden ambas presentarse en la misma disyunción elemental.

En general:

una tautología <u>no tiene</u> una FNC
y una contradicción <u>no tiene</u> ninguna FND.

Podemos resumir lo dicho en el siguiente esquema:

FND	$(p_1 \wedge q_1 \wedge \ldots) \vee (p_2 \wedge q_2 \wedge \ldots) \vee \ldots$	no hay contradicciones con esta forma.
FNC	$(p_1 \vee q_1 \vee \ldots) \wedge (p_2 \vee q_2 \vee \ldots) \wedge \ldots$	no hay tautologías con esta forma.

Hay dos formas de determinar una forma normal correspondiente a una fórmula dada.

Una es efectuar una serie de transformaciones basadas sobre equivalencias lógicas, tal como son dadas en la tabla 6.1

$$\begin{pmatrix} p \wedge p & eq & p \\ p \vee p & eq & p \end{pmatrix}$$

$$\begin{pmatrix} p \wedge (q \vee r) & eq & (p \wedge q) \vee (p \wedge r) \\ p \vee (q \wedge r) & eq & (p \vee q) \wedge (p \vee r) \end{pmatrix}$$

$$\sim \sim p \quad eq \quad p$$

$$\begin{pmatrix} \sim (p \vee q) & eq & \sim p \wedge \sim q \\ \sim (p \wedge q) & eq & \sim p \vee \sim q \end{pmatrix}$$

$$\begin{pmatrix} p \wedge (p \vee q) & eq & p \\ p \vee (p \wedge q) & eq & p \end{pmatrix}$$

(continúa tabla 6.1)

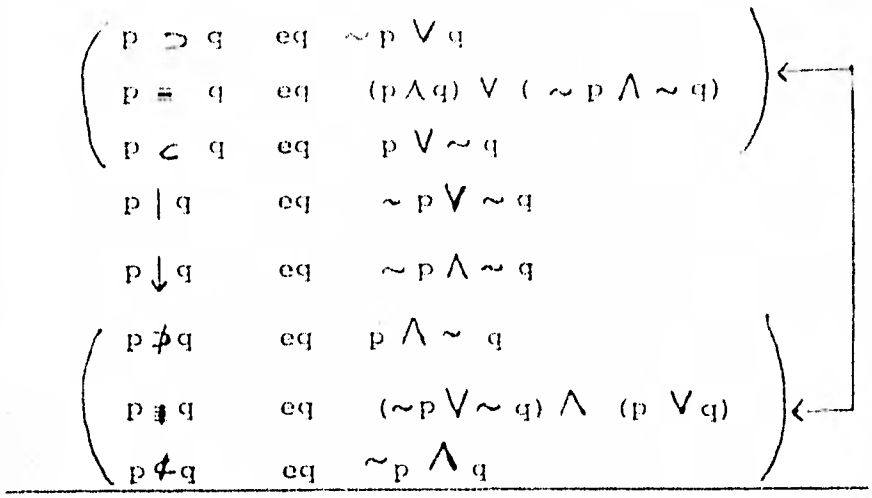


Tabla 6.1 Equivalencias lógicas básicas.

Ejemplo 6.1

$$\sim (p \wedge q) \supset \sim p \vee (p \mid \sim q)$$

eq. $\sim\sim(p \wedge q) \vee [\sim p \vee (\sim p \vee \sim\sim q)]$

eq. $(p \wedge q) \vee (\sim p \vee \sim p \vee q)$

eq. $(p \wedge q) \vee \sim p \vee q$ ... FND. Tres conjunciones elementales.

eq. $\sim p \vee q$ ... FND, dos conjunciones elementales

ó FNC, una disyunción elemental.

El otro método para obtener formas normales para una fórmula, consiste en el examen de la tabla de verdad. Es útil este método si no conocemos la fórmula sino sólo su tabla de verdad. Primeramente se obtiene la FNDP y luego se usa ésta pa

ra generar las otras formas normales. Así,

supongamos que A es una f.b.f. y que sabemos que, para una asignación particular de valores de verdad a sus variables, A toma el valor verdadero. Entonces, esa asignación debe corresponder a una u otra de las filas de la tabla de verdad de A para las que A tiene el valor V, y cualquiera fila que sea, todas las variables deben tener las asignaciones en esa fila. Y esto nos lleva a la -- FNDP.

Ejemplo 6.2.

Supongamos que tenemos la siguiente tabla de verdad y -- queremos determinar una fórmula que nos represente tal - tabla. Sólo conocemos la función por sus valores.

	p	q	r	f (p, q, r)	
	V	V	V	F	
	V	V	F	F	
3a.	V	F	V	V	←
	V	F	F	F	
5a.	F	V	V	V	←
6a.	F	V	F	V	←
	F	F	V	F	
8a.	F	F	F	V	←

indicadas para obtener una FNDP de f (p,q,r).

Fijándonos sólo en las filas que tienen V como valor de la - función, representamos rápidamente la función por una fórmula en FNDP. Para la función dada, esa fórmula es:

$$\underbrace{(p \wedge \sim q \wedge r)}_{\text{3a. fila}} \vee \underbrace{(\sim p \wedge q \wedge r)}_{\text{5a. fila}} \vee \underbrace{(\sim p \wedge q \wedge \sim r)}_{\text{6a. fila}} \vee \underbrace{(\sim p \wedge \sim q \wedge \sim r}_{\text{8a. fila}}$$

Por el uso de transformaciones apropiadas pueden derivarse diferentes formas para la función, como por ejemplo:

$$(\sim p \wedge (r \supset q)) \vee (p \wedge \sim q \wedge r)$$

ó

$$(\sim q \wedge (p \equiv r)) \vee (\sim p \wedge q).$$

Un detalle casi trivial pero importante es que ya teniendo la FNDP de una fórmula, es sencillo obtener la FNCP, de la siguiente manera:

PROCESO FNDP ——————→ FNCP

Sea A una f.b.f.

Notemos que $\sim A$ tiene el valor V, si y sólo si A tiene el valor F. De donde la FNDP para $\sim A$ se obtiene de la tabla de verdad para A, en la misma forma que la FNDP para A, excepto que son las filas con valores F las que se usan.

De esta forma podemos obtener la FNCP para A, usando las siguientes Leyes:

Leyes de De Morgan:

$$\sim(A \wedge B) \text{ eq } \sim A \vee \sim B$$
$$\sim(A \vee B) \text{ eq } \sim A \wedge \sim B$$

Ley de la Doble Negación: $\sim \sim A$ eq A.

Así, si la FNDP para $\sim A$ es $B_1 \vee B_2 \vee \ldots \vee B_n$,

entonces

$$A \text{ eq } \sim \sim A$$
$$\text{eq } \sim(B_1 \vee B_2 \vee \ldots \vee B_n)$$
$$\text{eq } \sim B_1 \wedge \sim B_2 \wedge \ldots \wedge \sim B_n.$$

Pero una típica B_i es $C_{i1} \wedge C_{i2} \wedge \ldots \wedge C_{in_i}$ de forma que $\sim B_i$ eq $\sim C_{i1} \vee \sim C_{i2} \vee \ldots \vee \sim C_{in_i}$ y así, mediante un uso cuidadoso de la Ley de doble negación, obtenemos la deseada FNCP para A.

Ejemplo 6.3

Consideremos la fórmula $(\sim p \supset r) \wedge (q \equiv p) \therefore A$.

La siguiente tabla muestra los valores de verdad para esta f.b.f.

p	q	r	$(\sim p \supset r)$	$\wedge$	$(q \equiv p)$
V	V	V	V	V	V
V	V	F	V	V	V
V	F	V	V	F	F
V	F	F	V	F	F
F	V	V	V	F	F
F	V	F	F	F	F
F	F	V	V	V	V
F	F	F	F	F	V

Como el valor es V para la 1a., la 2a. y la 7a. filas de esta tabla, la FNDP es la disyunción de las conjunciones elementales que representan estas filas.

$$\underbrace{(p \wedge q \wedge r)}_{\text{1a. fila}} \vee \underbrace{(p \wedge q \wedge \sim r)}_{\text{2a. fila}} \vee \underbrace{(\sim p \wedge \sim q \wedge r)}_{\text{7a. fila}} \therefore A \text{ en FNDP}$$

La FNDP para $\sim A$ se obtiene con las filas restantes:

$$(p \wedge \sim q \wedge r) \vee (p \wedge \sim q \wedge \sim r) \vee (\sim p \wedge q \wedge r) \vee$$
$$(\sim p \wedge q \wedge \sim r) \vee (\sim p \wedge \sim q \wedge \sim r) \therefore \sim A \text{ en FNDP}$$

Así pues, A eq $\sim(\sim A)$

eq $\sim(p \wedge \sim q \wedge r) \wedge \sim(p \wedge \sim q \wedge \sim r) \wedge$
$\sim(\sim p \wedge q \wedge r) \wedge \sim(\sim p \wedge q \wedge \sim r) \wedge$
$\sim(\sim p \wedge \sim q \wedge \sim r)$

eq $(\sim p \vee q \vee \sim r) \wedge (\sim p \vee q \vee r) \wedge (p \vee \sim q \vee \sim r)$
$\wedge (p \vee \sim q \vee r) \wedge (p \vee q \vee r)$.

Invirtiendo el orden de estas disyunciones elementales obtenemos la deseada FNCP de A.

Aunque estas dos formas canónicas se emplean para determinar si dos fórmulas son lógicamente equivalentes o no, un uso más frecuente es el de encontrar una fórmula más sencilla, lógicamente equivalente a la dada. Pero, para nuestros propósitos, el hecho más importante es que estas formas normales nos proporcionan un medio de generar una f.b.f. de un número arbitrario de variables, que tenga una tabla de verdad prescrita. Hemos dado, en efecto, un procedimiento por el que las operaciones de conjunción, disyunción y negación son suficientes para describir cualquier tabla de verdad. La fórmula que se obtiene por este procedimiento puede ser innecesariamente larga, pero tiene las propiedades deseadas y, en general, se puede simplificar.

7. NOTACION POLACA Y EL ARBOL DE UNA FORMULA.

En el sistema notacional usual, nos vemos limitados por la necesidad del uso de paréntesis y orden de precedencia, para preservar la claridad en la sintaxis de nuestras proposiciones. Esto es debido a que nuestra notación para las operaciones binarias es intercalada: cuando escribimos $A \supset B$, $A + B$ ó $A \Delta B$, debemos saber hasta dónde se extiende A a la izquierda

del operador y hasta dónde se extiende B a la derecha del operador. Al analizar el proceso de entender la sintaxis de una expresión, como debemos hacer para programar que una computadora "entienda" la expresión, es evidente que, para una expresión moderadamente compleja, esta comprensión contiene un número de búsquedas hacia adelante y hacia atrás en la expresión.

En 1951, el lógico polaco Jan Lukasiewicz sugirió que podía usarse una notación prefijada para todas las operaciones, de forma que podíamos escribir $\supset$ A B, + A B ó Δ A B; un uso consistente de una tal notación elimina toda necesidad de puntuación (uso de paréntesis). En esta sección examinamos esa notación y su relación con una cierta gráfica, a la que llamamos el "árbol" de una fórmula.

Aunque el contexto particular que usaremos es el cálculo proposicional, la notación es aplicable, siempre que nos encontramos o pueden definirse ecuaciones de n variables ($n \geq 1$).

Usaremos el siguiente simbolismo.

- Las proposiciones las denotaremos por letras minúsculas cursivas: p, q, r, s,...
- Los operadores y los conectivos por las siguientes letras mayúsculas:

N	negación
C	implicación
A	disyunción
K	conjunción
E	equivalencia.

la tabla 7.1 muestra nuestros propósitos.

Polaca	Estándar
Np	$\sim p$
Cpq	$p \supset q$
Apq	$p \vee q$
Kpq	$p \wedge q$
Epq	$p \equiv q$

Tabla 7.1. Relación entre las notaciones polaca y estándar.

- Usaremos también letras griegas minúsculas para representar a las fórmulas bien formadas (que debemos definir de nuevo, puesto que estamos usando un nuevo simbolismo).

Definición 7.1

Una fórmula bien formada (del cálculo proposicional y según la notación polaca) es una fórmula que puede obtenerse mediante un número finito de aplicaciones de las reglas siguientes:

1. Una variable proposicional aislada es una f.b.f.
2. Si α es una f.b.f., entonces Nα es una f.b.f.
3. Si α y β son f.b.f., entonces C$\alpha\beta$, A$\alpha\beta$, K$\alpha\beta$ y E$\alpha\beta$ son f.b.f.

Ejemplo 7.1

a) $(\sim p \supset r) \wedge (q \equiv p)$ en la notación polaca es KCNprEqp

b) $(\sim q \wedge (p \equiv r) \vee (\sim p \wedge q)$ se convierte en AKNqEprKNpq

Asignamos a nuestros símbolos RANGOS de la siguiente manera:

- Una variable proposicional tiene rango + 1, (ella sola constituye una f.b.f.).
- Un conectivo binario (C, A, K, E) tiene rango -1. (Reduce el número de partes de una sucesión naciendo de dos - de ellas una sola).
- Un conectivo singular (N) tiene rango 0. (No cambia el número de partes: p y Np constan ambos de una sola parte). Si tuviéramos conectivos ternarios, tendrían rango -2, y así sucesivamente.

El punto de vista en que estamos basando nuestra discusión es el de considerar los conectivos lógicos como si fueran exactamente eso: símbolos que sirven para conectar otras sucesiones de símbolos. Por ejemplo, p y q son dos símbolos no relacionados, y la sucesión pq consta de dos partes distintas, como- dos oraciones no separadas por signo de puntuación alguno. Pero C, como en Cpq, o K, como en Kpq, conecta estas dos partes y forma una sola sucesión unificada. Esto nos proporciona la base para nuestro algoritmo para fórmulas bien formadas.

ALGORITMO PARA F.B.F. POLACA

Sea $S = S_1 S_2 \ldots S_{n-1} S_n$ una fórmula en notación polaca, y sea- r_i el rango de S_i, $i = 1, 2, \ldots, n$.

Comenzando en el extremo derecho de la fórmula, fórmense las- sumas parciales:

$$\sum_n = r_n$$

$$\sum_{n-1} = r_{n-1} + r_n$$

$$\vdots$$

$$\sum'_2 = r_2 + \ldots + r_{n-1} + r_n$$

$$\sum'_1 = r_1 + r_2 + \ldots + r_{n-1} + r_n$$

Entonces S es una f.b.f. si y sólo si:

(1) toda $\sum'_i$ es positiva (≥ 1), $i = 1,\ldots,n$.

Y

(2) $\sum'_1 = 1$

Ejemplo 7.2.

(a)

Sucesión	K	C	N	p	r	F	q	p
Rangos	-1	-1	0	1	1	-1	1	1
Sumas	1	2	3	3	2	1	2	1

Aquí, por ejemplo, $\sum'_4 = 3$ lo que indica es que, en este punto, tenemos tres partes no conectadas de la fórmula, a saber, p, r y Eqp. La fórmula está bien formada.

(b)

Sucesión	A	K	N	p	C	r	q	K	p	K	N	q	r
Rangos	-1	-1	0	1	-1	1	1	-1	1	-1	0	1	1
Sumas	1	2	3	3	2	3	2	1	2	1	2	2	1

Como se verifican las condiciones (1) y (2) del algoritmo, es en efecto una f.b.f. en notación polaca esta sucesión de símbolos.

(c)

Sucesión	p	K	N	q	r
Rangos	1	-1	0	1	1
Sumas	2	1	2	2	1

Esta no es una f.b.f. (no se cumple (2)); tiene dos partes - sin conectar, p y KNqr. Es lo que indica $\sum_1 = 2$.

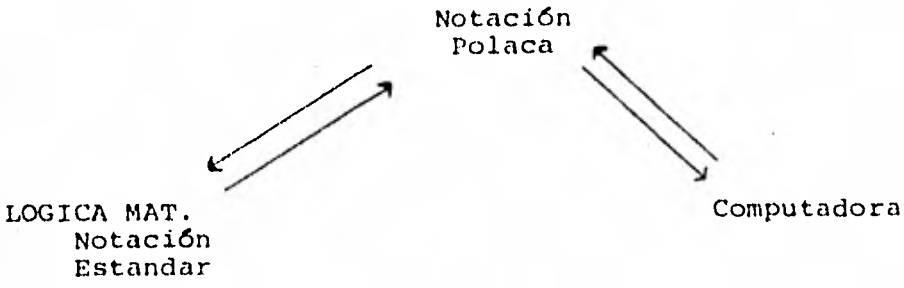
(d)

Sucesión	K	p	N	C	A	p	q
Rangos	-1	1	0	-1	-1	1	1
Sumas	0	1	0	0	1	2	1

Esta no es una f.b.f. (falla (2)). Falta un segundo "operando" para C. Escrita con la notación estandar, tiene la forma $p \wedge \sim ((p \vee q) \supset)$.

Es evidente la ventaja que hay de la notación polaca con la - estandar para comprobar fórmulas bien formadas, dándonos un - algoritmo que puede ser más fácilmente manejable por una computadora.

Es obvio que una f.b.f. en la notación estandar va a ser - - f.b.f. al traducirla a la notación polaca, y recíprocamente, - así que la notación polaca podemos utilizarla como un puente de comunicación entre el cálculo proposicional en notación estandar y una computadora; así como, por ejemplo, en sistemas numéricos, el sistema octal puede usarse como puente de conversión entre los sistemas binario y decimal, agilizando el - flujo de comunicación entre las bases 2 y 10.



Pasamos ahora a discutir lo que es el árbol de una fórmula. - Antes introducimos algunos conceptos de la teoría de gráficas.

Definición 7.2.

I Una gráfica es un par $\langle P, L \rangle$, donde:

- P es un conjunto de puntos, llamados "vértices".
- L es un conjunto de segmentos, llamados "lados",

tales que:

cada segmento de L tiene asociados con él a exactamente 2 puntos de P, que son sus vértices.

II Si a y b son vértices de una gráfica, entonces una cadena de a a b es un conjunto $C = \{ p_o, p_1, \ldots p_n \} \subseteq P$ de vértices tales que:

- $a = p_o$
- $b = p_n$

y

- para cada $i = 1, \ldots, n$, hay un lado $e_i \in L$ cuyos vértices son p_{i-1} y p_i.

III Una gráfica es conexa si hay una cadena de uno cualquiera de sus vértices a cualquier otro.

IV Un ciclo es una cadena $p_0, p_1, \ldots, p_n$; $n \geq 1$, - tal que $p_0, p_1, \ldots, p_{n-1}$ son distintos y $p_n = p_0$.

V Una gráfica conexa que no tiene ningún ciclo se llama un árbol.

Ejemplo 7.3

La figura 7.1 representa una gráfica cuyos vértices son- a, b, c, d, e, f y g y cuyos lados son ab, ac, bc, bd, - cf, de y ef.

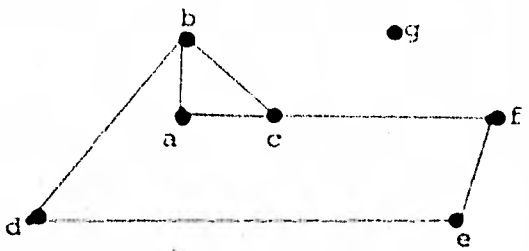


Fig. 7.1

Hay una cadena de a a f, por ejemplo $\{a, b, c, f\}$, o $\{a, c, f\}$ ó $\{a, b, d, e, d, b, c, f\}$.

La gráfica no es conexa, pero lo sería si prescindiera del punto g. Contiene varios ciclos, como $\{a, b, c, a\}$, $\{b, d, e, f, c, b\}$ y $\{a, b, d, e, f, c, a\}$.

Definición 7.3

Si α es una f.b.f. (en notación polaca), entonces el árbol de α es la gráfica definida por el siguiente proceso.

1. Si α es una sola variable proposicional, entonces el árbol de α es la gráfica con vértice α y sin lado - alguno.

2. Si α es de la forma Nβ, entonces el árbol de α es la gráfica cuyos vértices son $\{N\} \cup \{$vértices de $\beta\}$, y cuyos lados son los lados de β junto con un lado que une N y el conectivo principal de β (o el vértice de β, si β no tiene más que un vértice).

3. Si α es de la forma X$\beta\gamma$, donde X es uno de los conectivos C, A, E y D, entonces el árbol de α es la gráfica cuyos vértices son $\{X\} \cup \{$vértices de $\beta\} \cup \{$vértices de $\gamma\}$, y cuyos lados son los lados de β y de γ junto con un lado que une X y el conectivo principal de β, y un lado más que une X y el conectivo principal de γ.

Simbólicamente, podemos usar la representación que aparece en la figura 7.2, donde T (β) y T (γ) representan los árboles de β y γ, respectivamente.

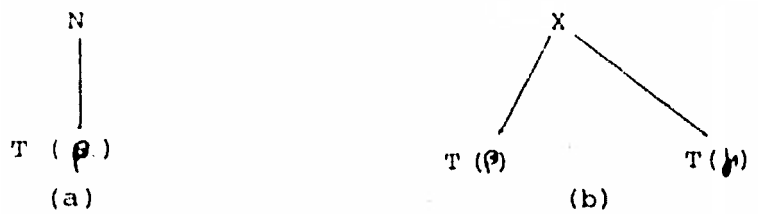


Fig. 7.2

(a) el árbol de Nβ

(b) el árbol de X$\beta\gamma$

Ejemplo 7.4.

Damos dos representaciones gráficas con las figuras 7.3- y 7.4.

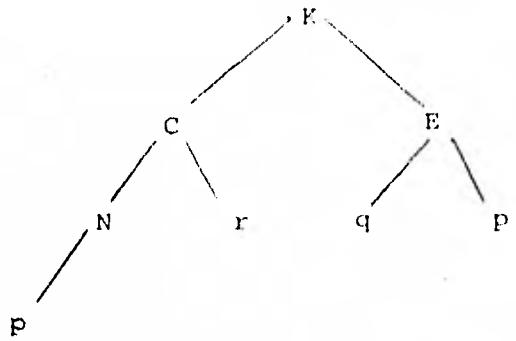


Fig. 7.3 Arbol de KCNprEqp

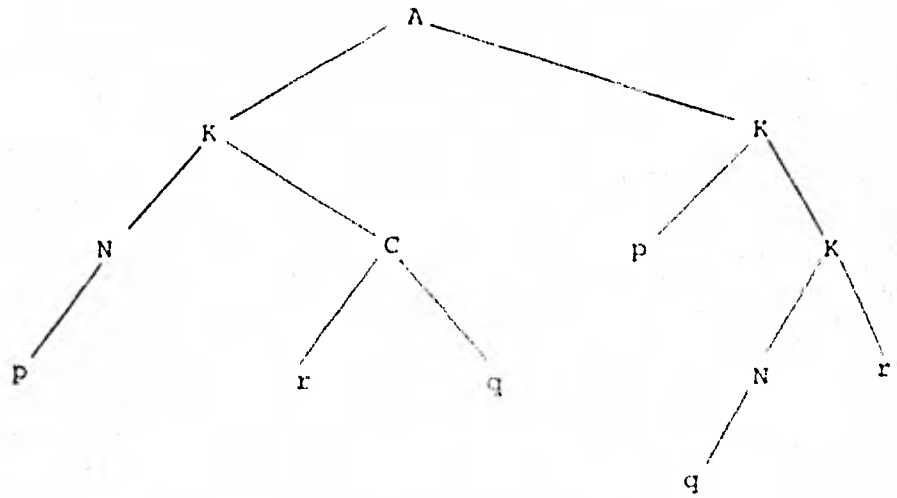


Fig. 7.4 Arbol de AKNpCrqKpKNqr.

Tres comentarios nos parecen oportunos.

1o. Los árboles que estamos usando son, en realidad, árboles "rotulados". Por ejemplo, la gráfica de la figura 7.5 puede ser el árbol de muchas fórmulas.

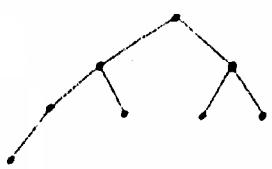


Fig. 7.5.

Cuando se rotulan los vértices, como en la Fig. 7.3, se convierte en el árbol de una fórmula particular.

2o. Como se tiene el hecho de que $p \supset q$ (Cpq) no es equivalente a $q \supset p$ (Cqp), el orden de izquierda a derecha de los vértices es importante para nuestros propósitos.

3o. El árbol de una fórmula podría haberse desarrollado perfectamente bien partiendo de la notación estándar de intercalaciones. El presente desarrollo se escogió a causa de la facilidad de construcción del árbol.

Si conocemos los símbolos que se presentan en cada nivel y el orden en que aparecen, es fácil construir el árbol.

Por ejemplo,

- Si el primer nivel contiene el símbolo K.
- El segundo los símbolos C, E.
- El tercero Nr q p
- El cuarto el símbolo p.

Podemos entonces construir el árbol de la figura 7.3.

Esta información se puede obtener fácilmente de la notación polaca para una fórmula proposicional, por medio del siguiente algoritmo "del árbol", que es una continuación del algoritmo para f.b.f. polaca dado antes.

ALGORITMO "DEL ARBOL" POLACO.

1. Fórmese la sucesión de sumas $\Sigma_1, \ldots, \Sigma_n$ como anteriormente en el algoritmo para f.b.f. Si la fórmula está bien formada, continúese con el paso siguiente.

2. Comenzando con el extremo izquierdo, fórmense las dos sucesiones $t_1,\ldots, t_n$ y $c_1,\ldots, c_n$, como sigue:

   (a) $t_1 = 1, \quad c_1 = 0$

   (b) para $i = 1,\ldots, n-1$

   Si $\sum_{i+1} > \sum_i$, hágase $t_{i+1} = t_i + 1$, $c_{i+1} = 0$

   Si $\sum_{i+1} = \sum_i$, hágase $t_{i+1} = t_i + 1$, $c_{i+1} = 1$

   Si $\sum_{i+1} < \sum_i$, determínese la última K ($\leq i$) tal que $c_{k=0}$; háganse entonces $t_{i+1} = t_k$,

   $c_{i+1} = 1, \quad c_k = 1.$

3. Si la fórmula dada está bien formada y el procedimiento anterior se ha seguido apropiadamente, entonces, al final -- $c_1 = 0$, $c_i = 1$ $(i = 2,\ldots,n)$ y t_i es el nivel del i-ésimo-símbolo en el árbol, según su orden de aparición.

Ejemplo 7.5.

Ilustramos la aplicación de este algoritmo con las fórmulas del ejemplo 7.4, KCNprEqp y AKNpCrqKpKNqr, figuras 7.3 y 7.4 respectivamente.

(a)

Fórmula	K	C	N	p	r	E	q	p
Rangos	-1	-1	0	1	1	-1	1	1
$\sum_i$	1	2	3	3	2	1	2	1
t	1	2	3	4	3	2	3	3
c	0	0	0	1	1	1	0	1
		1	1				1	

Entonces, según nuestro algoritmo, tenemos:

en el primer nivel: K

en el segundo nivel: C, E en ese orden.

en el tercer nivel: N, r, q, p en ese orden.

en el cuarto nivel: p

Hacemos la observación que, tal como estamos disponiendo el dibujo de nuestros árboles, el orden de aparición es el siguiente:

de los niveles: de arriba-abajo ↓ 1o. 2o. 3o. ⋮

de las letras: de izquierda a derecha ⟶

Así, repitiendo la figura 7.3, tenemos:

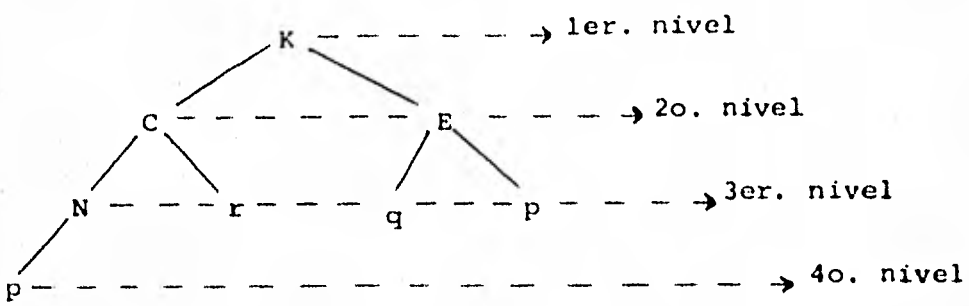


Fig. 7.3. (bis)

(b) Fórmula	A	K	N	p	C	r	q	K	p	k	N	q	r
Rangos	-1	-1	0	1	-1	1	1	-1	1	-1	0	1	1
$\sum$	1	2	3	3	2	3	2	1	2	1	2	2	1
t	1	2	3	4	3	4	4	2	3	3	4	5	4
c	0	0	0	1	1	0	1	1	0	1	0	1	1
		1	1			1			1		1		

Con esto tenemos la distribución de la siguiente manera:

1er. nivel: A

2o. nivel: K, K

3er. nivel: N, C, p, K

4o. nivel: p, r, q, N, r

5o. nivel: q

Con lo cual reproducimos rápidamente la figura 7.4, árbol de-la fórmula dada.

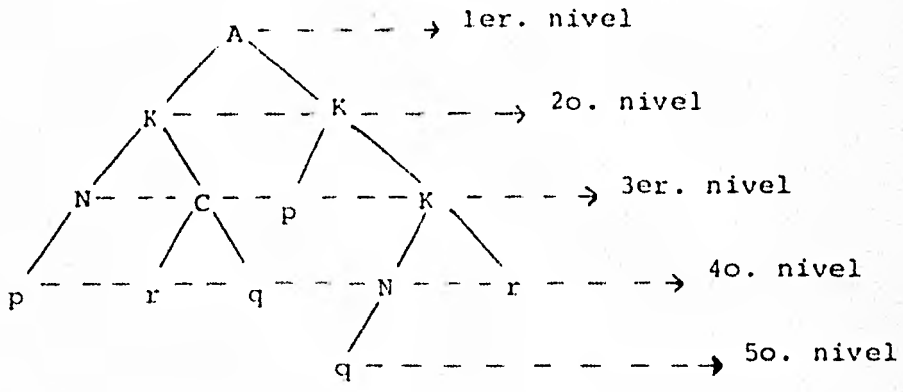


Fig. 7.4. (bis)

Revisando la anterior discusión, vemos que en realidad el árbol de una fórmula en notación polaca no es más que otra manera de escribir la fórmula misma, es una forma geométrica bastante bella, que al dibujarla se pone de manifiesto el significado de la aparentemente abstrusa notación polaca. Hemos dado un algoritmo para dibujar el árbol de una fórmula dada, es bastante obvio que el proceso recíproco no ofrece dificultades técnicas, o sea: dado un árbol, escribir la fórmula correspondiente; es un proceso trivial que consiste más o menos en desarmar nuestro árbol por las ramas, juntar los troncos en un manojo y echar todo en un cesto.

Para finalizar esta sección, manifestamos tres hechos claros del tema tratado:

1o. La notación polaca para una fórmula está muy íntimamente relacionada con la estructura de árbol. En realidad, si examinamos las letras en sucesión, comenzando al extremo izquierdo de la fórmula, lo que en realidad estamos haciendo exactamente es trazar el árbol de la fórmula, escogiendo en cada paso la rama no trazada que está más a la izquierda.

2o. La utilidad de la notación polaca para el trabajo computacional es evidente: si procedemos de derecha a izquierda de una fórmula, entonces no encontramos ningún operador ni conectivo antes de que conozcamos su operando. No necesitamos por ello retrasar un cálculo porque no tengamos todavía los datos.

Compárese, por ejemplo, el cálculo de:

(3+4)/ (5- (2x (1+3))) x (2- (4x5)) / (3+ (1x2)-4) +2

(partiendo de cualquiera de los extremos) con el de:

AMDA34S5M2A13DS2M45SA3M1242

(desde el extremo derecho), donde A, S, M y D denotan la-

adición, la sustracción, la multiplicación y la división, respectivamente.

3o. La falta de puntuación (paréntesis) depende de tener símbolos distintos. Siempre que vemos "13" debemos saber si éste denota el número trece o los dos números 1 y 3. Si los símbolos no son precisos, entonces debemos reintroducir la puntuación, al menos hasta el punto de tener algún delimitador (quizá un blanco) para separar símbolos.

8. CONJUNTOS MINIMALES DE CONECTIVOS.

El problema de que vamos a ocuparnos en esta sección es el de cuántos conectivos necesitamos especificar realmente para desarrollar nuestra lógica. Aunque esto es de interés teórico, la contestación a este problema tiene también implicaciones prácticas, pues si estamos diseñando una máquina lógica, nos gustaría estandarizar los elementos básicos de la máquina. Parece que no sería conveniente usar diez tipos de elementos básicos, si fueran suficientes tan sólo dos o tres.

Por otra parte, hay una cierta pérdida al usar menos elementos: las expresiones lógicas y, por tanto, los circuitos, que representan funciones particulares, pueden hacerse más complicados. Así pues, en los problemas de diseño práctico debemos balancear cuidadosamente el número de tipos diferentes de elementos básicos y los circuitos necesarios para usar estos elementos. Con esto vemos que, en la práctica, lo mínimo no siempre es lo mejor, al igual que tampoco lo máximo.

Como cualquier función puede representarse por una expresión en FND, es claro que, cuando mucho, los conectivos que necesitamos son la conjunción, la disyunción y la negación.

Por otra parte, las Leyes de DeMorgan nos permiten conjugar-tales conectivos,

$$(p \vee q) \text{ eq. } \sim(\sim p \wedge \sim q)$$
$$(p \wedge q) \text{ eq. } \sim(\sim p \vee \sim q)$$

luego, lo que necesitamos realmente tener a nuestra disposi--ción es la negación y: o la conjunción o la disyunción. Bueno, ¿se podría hacer aún mejor?, veamos:

Trabajemos con la disyunción y la negación.

- Como la negación es una operación que involucra solamente un operando, no podemos eliminar la disyunción: de otro - modo, no tendríamos manera de expresar funciones de más - de una variable.

- Su pudiésemos representar la negación por una expresión - disyuntiva, esa expresión incluiría sólo a una variable - proposicional, puesto que la negación es una función singular, de una sola variable. Así, $\sim p$ tendría que ser - expresado por $p \vee p$, $(p \vee p) \vee p$, $(p \vee p) \vee (p \vee p)$, o alguna - - otra expresión de ese tipo. Pero podemos ver rápidamente que una expresión semejante es lógicamente equivalente a- p; de donde se deduce que no hay expresión disyuntiva alguna que represente a $\sim p$, y, por lo tanto, tenemos que- retener la $\sim$.

- Luego, la función lógica puede expresarse en términos de- disyunción y negación, y hay funciones que no pueden expresarse usando solamente una de ellas.

Decimos entonces que la negación y la disyunción, $\{\sim, \vee\}$, - forman UN CONJUNTO MINIMAL DE CONECTIVOS para el cálculo proposicional.

Análogamente, la negación y la conjunción, $\{\sim, \wedge\}$, forman también un conjunto minimal de conectivos.

Es posible seleccionar también otros conjuntos minimales de conectivos. Por ejemplo, el trazo de Sheffer, $|$, constituye, él solo, un conjunto minimal de conectivos, lo mismo que la flecha de Pierce $\downarrow$.

Para mostrar que el trazo constituye un conjunto minimal necesitamos simplemente probar que tanto la negación (o la conjunción, pues ya sabemos que $\{\sim, \vee\}$ y $\{\sim, \wedge\}$ son minimales) son expresables en términos del trazo, puesto que, entonces, por lo dicho, cualquier función será también expresable. Por inspección de las tablas se verifica fácilmente que:

$$\sim p \text{ eq } p|p$$

$$p \vee q \text{ eq } (p|p) \mid (q|q)$$

También $p \wedge q$ eq $(p|q) \mid (p|q)$

lo cual prueba que $\{|\}$ es minimal.

De manera análoga:

$$\sim p \text{ eq } p \downarrow p$$

$$p \vee q \text{ eq } (p \downarrow q) \downarrow (p \downarrow q)$$

o bien $p \wedge q$ eq $(p \downarrow p) \downarrow (q \downarrow q)$

prueba que $\{\downarrow\}$ es minimal.

También, para concluir con uno más,

$$\sim p \text{ eq} \sim p$$

$$p \vee q \text{ eq } \sim p \supset q$$

o bien $p \wedge q$ eq $\sim (p \supset \sim q)$

prueba que $\{\sim, \supset\}$ es minimal.

Resumimos lo obtenido en la tabla 8.1 que muestra algunos conjuntos minimales de conectivos para el cálculo proposicional.

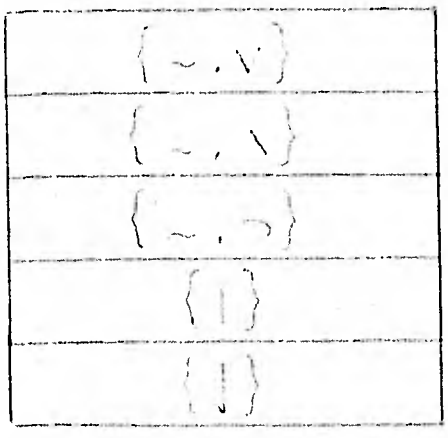


Tabla 8.1

9. UN ENFOQUE AXIOMATICO.

Como ya comentamos antes, el cálculo proposicional es una forma de calcular con proposiciones. Hasta el momento nuestros cálculos al respecto se han limitado a los de tablas de verdad y proposiciones lógicamente equivalentes. Ahora planteamos la cuestión bajo un nuevo enfoque y mostrar cómo pueden derivarse nuevas expresiones partiendo de otras dadas. El enfoque es formal, y nos da un procedimiento para la formación de nuevas sucesiones de cadenas de símbolos, partiendo de sucesiones dadas. Por el momento, el concepto de verdad no entra en nuestro cuadro; aunque más tarde veremos que las sucesiones que derivamos de nuestros axiomas son, en realidad, tautologías.

- Las sucesiones de símbolos que usaremos serán las formadas por las variables proposicionales, los conectivos y los paréntesis, como hemos estado haciendo, salvo el hecho de que sólo consideraremos como conectivos "legítimos" la negación y la condicional. Las expresiones en que aparezcan otros conectivos tan sólo las usaremos como abreviaturas de expresiones en estos dos conectivos.

- Salvo por lo dicho, las reglas para definir fórmulas bien formadas son las dadas en la sección 2.

El proceso de la generación de sucesiones comienza con la elección de un conjunto no vacío de f.b.f. al que llamamos el conjunto de los axiomas, y constituyen la base para la generación: las sucesiones que generamos se construyen partiendo de los axiomas. Así, la elección del conjunto de axiomas afecta materialmente al conjunto de sucesiones que se generan. La elección puede ser muy personal. Nosotros escogemos el siguiente conjunto de tres axiomas:

AXIOMAS

Axioma 1. $p \supset (q \supset p)$

Axioma 2 $(p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))$

Axioma 3. $(\sim p \supset \sim q) \supset ((\sim p \supset q) \supset p)$

Una vez escogido un conjunto de axiomas, el siguiente paso consiste en especificar un conjunto de reglas mediante las cuales vamos a generar las nuevas sucesiones. Nosotros escogemos las dos siguientes:

REGLAS DE INFERENCIA.

Modus Ponens. Dadas las sucesiones A y $A \supset B$, está permitido generar la sucesión B.

Substitución. Dada la sucesión A, la f.b.f. B, y una variable proposicional, digamos p, está permitido generar una sucesión C, substituyendo por B cada aparición de p en A.

Comentario:

En algunas presentaciones, el uso de la Regla de Substitución se evita por la elección de un conjunto infinito de axiomas que están representados por un esquema de axiomas.

Por ejemplo, en lugar de nuestro Axioma 1 tendríamos el esquema de axioma 1: $A \supset (B \supset A)$, donde A y B denotan f.b.f.; este esquema de axiomas denotaría entonces el conjunto de todas aquellas fórmulas que constituirían un subconjunto del conjunto de axiomas.

En nuestra presentación, las fórmulas, en este conjunto, deben generarse por substitución en el axioma dado.

Definición 9.1.

Una f.b.f. A es un teorema, si existe una sucesión finita de f.b.f. $A_1, A_2, \ldots, A_n$ tales que:

(1) cada A_i es: o un axioma, o está derivada de fórmulas precedentes de la lista por una de las reglas de inferencia. y

(2) An es la f.b.f.A.

Notas:

- Una sucesión tal se llama prueba, en particular una prueba de A.
- Así pues, A es teorema si hay una prueba de A.
- En particular, un Axioma es un Teorema, ya que existe una prueba de él, a saber, el mismo axioma.
- La notación que usaremos es $\vdash$ A para indicar que A es teorema.
- Nótese que esta definición nos da una formulación precisa de los conceptos de "teorema" y "prueba" dentro de la teoría.

Ilustramos el proceso de la generación de sucesiones probando un teorema.

Teorema 9.1.

$$\vdash ((q \supset r)) \supset ((p \supset q) \supset (p \supset r))$$

Prueba.

(1) $(p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))$ Ax. 2

(2) $p \supset (q \supset p)$ Ax.1

(3) $((p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r)))$

$\supset ((q \supset r) \supset ((p \supset (q \supset r)) \supset ((p \supset q) \supset$

$(p \supset r))))$ Sust. (2)

(4) $(q \supset r) \supset ((p \supset (q \supset r))$

$\supset ((p \supset q) \supset (p \supset r)))$ M.P. (1), (3)

(5) $(q \supset r) \supset (p \supset (q \supset r))$ Sust. 2

(6) $((q \supset r) \supset ((p \supset (q \supset r))$

$\supset ((p \supset q) \supset (p \supset r))))$

$\supset (((q \supset r) \supset (p \supset (q \supset r)))$

$\supset ((q \supset r) \supset ((p \supset q)$

$\supset (p \supset r))))$ Sust. (1)

(7) $((q \supset r) \supset (p \supset (q \supset r)))$

$\supset ((q \supset r) \supset ((p \supset q)$

$\supset (p \supset r)))$ M.P. (4), (6).

(8) $((q \supset r)) \supset ((p \supset q) \supset (p \supset r))$ M.P. (5), (7)

Donde "Ax", significa Axioma, "M.P.", Modus Ponens y "Sust"., substitución.

Definición 9.2

- Decimos que una f.b.f. Wj depende inmediatamente de Wi - si Wj se deriva de Wi por substitución;
  y que:

- W_k depende inmediatamente de Wi y Wj si W_k se deriva de Wi y Wj por Modus Ponens.

- Decimos que W_k depende de Wi si hay una sucesión de f.b.f. $W_{i_1} = W_i, \ldots, W_{i_n} = W_k$, tal que cada Wj en la sucesión depende inmediatamente de las fórmulas precedentes en la sucesión.

Podemos representar gráficamente una prueba como un árbol en- que se muestran las relaciones de dependencia entre las fórmulas en la prueba. Las "hojas" del árbol son ocurrencias de los axiomas, y la "raíz" es el Teorema..

Ejemplo 9.1

Hacemos un diagrama de la prueba del teorema 9.1 como un árbol en donde cada nodo está marcado con el número del renglón correspondiente en la prueba, lo que se muestra en la figura 9.1

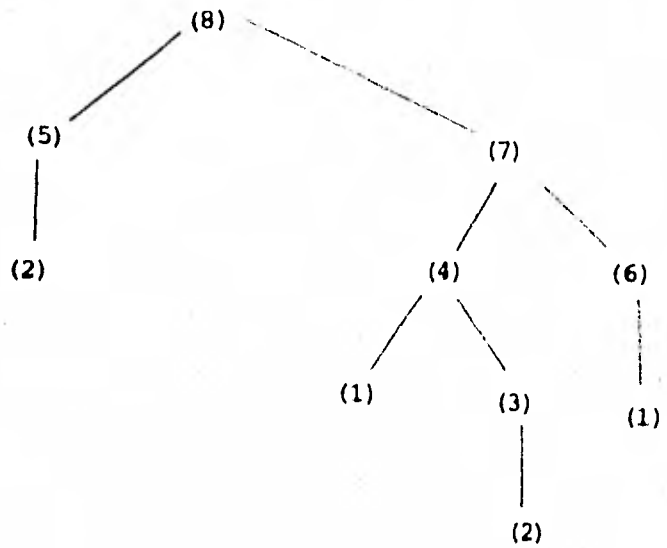


Figura 9.1

Vemos ahora las relaciones del sistema que estamos estudiando con las funciones de verdad. Desde el punto de vista axiomático, sería agradablemente "perfecto" que todo lo que se puede probar es cierto y viceversa. Para el cálculo proposicional éste es ciertamente el caso.

- Checando rápidamente por tablas de verdad, vemos que todos nuestros axiomas son tautologías;

- Por el ejemplo 5.1, el Modos Ponens preserva las tautologías. Es decir, si $\models A$ y $\models A \supset B$, entonces también $\models B$.

- Además, como cualquier substitución se hace en todas las apariciones de una variable proposicional, las tautologías se preservan también bajo esta regla de inferencia.

Por tanto, cualquier teorema es una tautología.

La afirmación recíproca - cualquier tautología es un teorema- es también cierta y se conoce como el "teorema de Completez"- del Cálculo Proposicional. Omito la prueba. Esta puede verse en Introduction to Mathematical Logic - E. Mendelson - pág. 37 Edit. Van Nostrand - 1979-.

Definición 9.3

Sea $\{A_1, A_2, \ldots, A_n, B\}$ un conjunto finito de f.b.f.

Decimos que B es una consecuencia de, o que es deducible de, - $A_1, A_2, \ldots, A_n$ (lo que escribimos como $A_1, A_2, \ldots, A_n \vdash B$), si - existe una sucesión finita de f.b.f., $P_1, P_2, \ldots, P_k$, tales - que:

(1) cada P_i es:

- o un axioma,
- o una de las hipótesis $A_1, \ldots, A_n$
- o se deriva de las fórmulas precedentes en la sucesión por una de las reglas de inferencia.

(2) ninguna fórmula P_i se deriva por substitución de una variable que aparece en una fórmula de la que depende P_i y

(3) P_k es la f.b.f. B.

NOTAS:

- a la sucesión. $P_1,\ldots,P_k$ se le llama una deducción de B a partir de las hipótesis $A_1,\ldots,A_n$.
- Si B es un teorema, es consecuencia del conjunto vacío - de hipótesis, y escribimos entonces: $\vdash B$.
- De hecho, una prueba de un teorema, es una deducción del teorema a partir de ninguna hipótesis.

Vemos ahora el objetivo para lo cual dimos la definición anterior.

EL TEOREMA DE LA DEDUCCION.

Sea Γ un conjunto de f.b.f. y sean

A y B f.b.f.

Si $\Gamma, A \vdash B$, entonces $\Gamma \vdash A \supset B$.

En particular: Si $A \vdash B$, entonces $\vdash A \supset B$.

Este teorema fué probado por primera vez por Herbrand en 1930. Aquí omito la prueba, pero la idea de la misma es la siguiente: "buscar una deducción de B partiendo de Γ, A y reemplazar cada P_i de la deducción por $A \supset P_i$." Lo que resulta ya no es una - deducción, pero, por la interpolación de fórmulas adicionales- adecuadas, se convierte en una deducción de $A \supset B$ partiendo de- la hipótesis .

El Teorema de la deducción es un instrumento inapreciable, - - pues nos permite escribir rápidamente pruebas de teoremas que, de otra forma, resultarían muy complicadas.

Como ejemplo de su aplicación, veamos cómo se simplifica la -

prueba de nuestro teorema 9.1.

Ejemplo 9.2.

Volvemos a probar el Teo. 9.1 tomando como hipótesis: $q \supset r$, $p \supset q$ y p.

(1)	p	Hip.
(2)	$p \supset q$	Hip.
(3)	q	M.P. (1), (2)
(4)	$q \supset r$	Hip.
(5)	r	M.P. (3), (4).

Tenemos pues una deducción $q \supset r$, $p \supset q$, $p \vdash r$. Aplicando el Teo. de la Deducción tres veces, obtenemos sucesivamente:

$$q \supset r,\ p \supset q \vdash p \supset r$$
$$q \supset r \vdash (p \supset q) \supset (p \supset r)$$
$$\vdash (q \supset r) \supset ((p \supset q) \supset (p \supset r))$$

Lo que ahora tenemos no es una prueba del teo. 9.1 de acuerdo a como hemos definido "prueba" dentro de nuestra teoría. Pero las técnicas usadas en probar el Teo. de la Deducción son constructivas y, mediante su uso, podemos transformar la deducción que hemos escrito en una prueba formal del teorema. Es inusual hacerlo, salvo en el adiestramiento escolar, pero no pasa de ser un proceso ya mecánico.

Así pues, tenemos aquí la contrapartida de la relación entre consecuencia lógica y tautología que mencionamos en la sección 5. Esta correspondencia y la correspondencia entre tautologías y teoremas hace trivial una gran parte del trabajo -

en el cálculo proposicional.

- ¿Es una f.b.f. un teorema?. Para encontrar la respuesta sólo necesitamos ver su tabla de verdad -un proceso mecánico.

- Si es un teorema, ¿qué hay de su prueba?. Reemplazamos primero el problema por el de encontrar una deducción -- adecuada. Ya encontrada, otro proceso mecánico transforma la deducción en la prueba deseada.

  La única parte difícil que resta es la de encontrar una deducción adecuada; para esta tarea, pocas son las guías conocidas que se encuentran entre el trabajo mecánico en un extremo y la intuición altamente desarrollada en el otro.

b) VECTORES BINARIOS.

1. Conjuntos.
2. Lógica.
3. Números.

COMENTARIO.

Ante una computadora, uno como principiante puede sentir extrañeza porque la computadora, supuestamente un dispositivo numérico, pueda "leer" caracteres alfabéticos. En esta parte del capítulo se trata de señalar la naturaleza abstracta de las representaciones simbólicas internas de una computadora, examinando varias interpretaciones posibles de un vector binario, dando lugar tales interpretaciones a un sistema lógico relacionado con nuestra lógica básica que hasta ahora hemos desarrollado.

Queremos examinar una estructura particular y ver cómo puede interpretarse de varios modos. Esta estructura es un vector o n-tuple ordenado de ceros y unos. Consideraremos los cuatro vectores siguientes, en todo lo que a continuación diremos, como nuestros ejemplos:

$$V_1 = \langle 0, 1, 1, 0, 1, 0 \rangle \quad o \quad 0\ 1\ 1\ 0\ 1\ 0$$
$$V_2 = \langle 1, 0, 1, 1, 0, 1 \rangle \quad o \quad 1\ 0\ 1\ 1\ 0\ 1$$
$$V_3 = \langle 1, 0, 0, 1, 1, 0 \rangle \quad o \quad 1\ 0\ 0\ 1\ 1\ 0$$
$$V_4 = \langle 0, 1, 1, 0, 0, 1 \rangle \quad o \quad 0\ 1\ 1\ 0\ 0\ 1$$

1. Conjuntos.

Es posible usar vectores para representar conjuntos, conviniendo en que cada componente del vector denote un elemento particular de nuestro universo. Usamos un "1" para indicar el hecho de que el elemento representado por una componente dada está en el conjunto considerado, y un "0" para indicar que no está. Este vector puede pensarse como la función característica del conjunto, para casos finitos.

Por ejemplo, si nuestro universo es $U = \{1, 2, 3, 4, 5, 6\}$ y convenimos que la i-ésima componente represente al elemen-

to 1, entonces nuestros vectores representarán en forma res-- pectiva a los siguientes conjuntos:

V_1 a A = { 2, 3, 5 }

V_2 a B = { 1, 3, 4, 6 }

V_3 a C = { 1, 4, 5 }

V_4 a D = { 2, 3, 6 }

Bajo este "truco" de representación, las distintas operaciones de conjuntos se efectúan más fácilmente, comparando las representaciones del vector por la presencia o ausencia de unos. Así, digamos, la unión de conjuntos requiere la presencia de un 1 en una posición dada en cualquiera de los vectores, mientras que la intersección de conjuntos exige la presencia de un 1 en una posición dada en ambos vectores.

Ejemplo 1.1

A: 0 1 1 0 1 0

B: 1 0 1 1 0 1

$A \cup B$: 1 1 1 1 1 1 (1 en A o en B); $A \cup B = U$

$A \cap B$: 0 0 1 0 0 0 (1 tanto en A como en B); $A \cap B = \{ 3 \}$

$A \Delta B$: 1 1 0 1 1 1 (1 en sólo uno de los dos); $A \Delta B = \{1,2,4,5,$

A^c: 1 0 0 1 0 1 (se intercambian 0 y 1).

Nótese que esta representación es aplicable en caso de ser finitos los conjuntos. Qué tan finitos, no importa mucho, pues para efecto de tamaño, lo que se requiere es memoria (en la computadora) pero el algoritmo de las operaciones sigue siendo el mismo, y la complejidad de las mismas se resuelve de una manera automática (por la máquina) al especificar cuál es

nuestra "convención" para la representación (codificación).

2. L O G I C A.

Supongamos ahora que tenemos un conjunto de objetos codificados de acuerdo con la presencia o ausencia de cierto número de propiedades -seis en nuestros ejemplos-. Podemos usar en nuestro código un "1" para indicar la presencia de una propiedad, y un "0" para indicar la ausencia de ella.

En otras palabras, el 1 y el 0 toman ahora el lugar de las letras V y F, respectivamente, para la sentencia: "Este objeto tiene la propiedad P_j".

Por ejemplo, en problemas de catalogación o clasificación, los objetos pueden ser personas, y las propiedades, el entrenamiento para el manejo de ciertas máquinas, la posibilidad de estar disponibles a ciertos tiempos, la voluntad de trabajo, etc. En trabajos de recuperación de información, al aplicarse a un documento, el 1 y el 0 podrían indicar la presencia o ausencia de ciertas palabras claves en el documento; aplicado a una petición, los mismos 1 o 0 indicarían un interés o una falta de interés en esas palabras clave.

Podemos entonces determinar varias combinaciones de estas propiedades, aplicándolas a las operaciones lógicas.

Ejemplo 2.1

Los objetos: son personas

Las propiedades: son los tiempos en que están disponibles.

¿Cuándo pueden encontrarse W_2 y W_3?.

Hacemos el análisis:

	t_1	t_2	t_3	t_4	t_5	t_6
V_2 :	1	0	1	1	0	1
V_3 :	1	0	0	1	1	0
$V_2 \wedge V_3$:	1	0	0	1	0	0

Así pues, V_2 y V_3 pueden encontrarse en cualquiera de los períodos t_1 o t_4.

Ejemplo 2.2

Los objetos: son empleados de vigilancia.

Las propiedades: son tiempos en que están de guardia.

¿Cuándo está al menos uno de los dos vigilantes V_1 o V_3 de guardia?.

V_1 :	0	1	1	0	1	0
V_3 :	1	0	0	1	1	0
$V_1 \vee V_3$:	1	1	1	1	1	0

En todo momento, salvo en el lapso t_6 está de guardia uno de los afectados.

¿Qué combinaciones de vigilantes cubrirían por completo todo el tiempo?

Observando V_1, V_2, V_3 y V_4, formamos las combinaciones, digamos, de parejas de vigilantes, vemos que:

$V_1 \vee V_2$	:	1	1	1	1	1	1
$V_1 \vee V_3$	:	1	1	1	1	1	0
$V_1 \vee V_4$	:	0	1	1	0	1	1
$V_2 \vee V_3$	:	1	0	1	1	1	1
$V_2 \vee V_4$	:	1	1	1	1	0	1
$V_3 \vee V_4$	:	1	1	1	1	1	1

Así pues, sólo las parejas V_1 —— V_2 y V_3 —— V_4 cubrirían- por completo todo el tiempo.

Ejemplo 2.3

Los objetos: son documentos.

Las propiedades: palabras clave.

Supongamos que estamos buscando documentos en los que se en-- cuentren las palabras clave P_1 y P_4. Es decir, necesitamos - un 1 en las posiciones primera y cuarta del vector, pero no - nos preocupa qué es lo que aparezca en las otras posiciones.- Esto puede comprobarse usando el condicional, de la manera si guiente:

Fórmese el vector con la propiedad P = 1 0 0 1 0 0 y comprué bese:

P	:	1	0	0	1	0	0
V_1	:	0	1	1	0	1	0
$P \supset V_1$	:	0	1	1	0	1	1

P	:	1	0	0	1	0	0
V_2	:	1	0	1	1	0	1
$P \supset V_2$	:	1	1	1	1	1	1

P	:	1	0	0	1	0	0
V_3	:	1	0	0	1	1	0
$P \supset V_3$	:	1	1	1	1	1	1

P	:	1	0	0	1	0	0
V_4	:	0	1	1	0	0	1
$P \supset V_4$	:	0	1	1	0	1	1

Así vemos que los documentos deseados son aquéllos que tienen unos en ambas posiciones, en este caso V_2 y V_3.

Supongamos ahora que los que necesitamos son aquellos documentos con las palabras claves P_1 y P_4 pero no P_6.

Formamos el vector con propiedad Q = 1 1 1 1 1 0 y comparamos con V_2 y V_3, usando de nuevo la condicional:

V_2	:	1	0	1	1	0	1	V_3	:	1	0	0	1	1	0
Q	:	1	1	1	1	1	0	Q	:	1	1	1	1	1	0
$V_2 \supset Q$	:	1	1	1	1	1	0	$V_3 \supset Q$	:	1	1	1	1	1	1

tenemos pues que V_3 es el documento que estábamos buscando.

Nótese la manera sencilla en que se resolvió el problema, con sólo cambiar el orden del condicional poniendo una propiedad adecuada y utilizando el resultado anterior.

3. NUMEROS.

La notación numérica ordinaria en uso corriente es una notación posicional o vectorial con una base de 10. Así, por ejemplo, 1, 473 denota el número $1 \times 10^3 + 4 \times 10^2 + 7 \times 10^1 + 3 \times 10^0$.

Como un resultado del análisis matemático tenemos que podemos usar el mismo tipo de notación con cualquier entero positivo r como base, usando solamente los "dígitos" 0, 1,..., r-1. Por ejemplo, en el sistema octal (base 8), 1,473 representaría al número $1 \times 8^3 + 4 \times 8^2 + 7 \times 8^1 + 3 \times 8^0$ que es 848 en el sistema decimal.

En particular, estamos interesados en el uso de la base 2. En

este caso, los únicos dígitos usados son 0 y 1. Así, por - - ejemplo, viendo nuestros cuatro vectores de antes como numera les binarios, tendrían la siguiente equivalencia en el siste- ma decimal:

$$V_1: \quad 0\ 1\ 1\ 0\ 1\ 0_{(2)} = 26_{(10)}$$

$$V_2: \quad 1\ 0\ 1\ 1\ 0\ 1_{(2)} = 45_{(10)}$$

$$V_3: \quad 1\ 0\ 0\ 1\ 1\ 0_{(2)} = 38_{(10)}$$

$$V_4: \quad 0\ 1\ 1\ 0\ 0\ 1_{(2)} = 25_{(10)}$$

donde el subíndice indica la base del numeral.

Nótese que estamos usando la convención que permite a los numerales binarios comenzar con un cero, de forma que podemos - escribir, digamos el tres, como 11 (= 1 x 2' + 1 x 2°) o como 011 (= 0 x 2^2 + 1 x 2^1 +1 x 2°). Esto nos permite usar vecto res de longitud fija, y es un convenio en uso corriente entre los usuarios de computadoras.

ARITMETICA BINARIA.

La aritmética en el sistema de base dos es bastante sencilla. Como ya sabemos, tenemos dos símbolos (0 y 1) y dos operado-- res, + y X, cuyas reglas están dadas por las tablas 3.1 y 3.2 respectivamente.

+	0	1
0	0	1
1	1	10

Tabla 3.1

X	0	1
0	0	0
1	0	1

Tabla 3.2

En la tabla 3.1 nótese que 1 + 1 = 2 decimal, o sea 10 en binario.

Ejemplo 3.1

V_1 : 0 1 1 0 1 0 — 26

V_2 : 1 0 1 1 0 1 — + 45

$V_1 + V_2$: 10 0 0 1 1 1 — 71

V_1 : 0 1 1 0 1 0

V_4 : 0 1 1 0 0 1

0 1 1 0 1 0

0 1 1 0 1 0

0 1 1 0 1 0

$V_1 \times V_4$: 1 0 1 0 0 0 1 0 1 0 (26 X 25 = 650)

Esta aritmética está también íntimamente conectada con nuestro sistema lógico, como puede verse al examinar la adición con más detalle. Cuando añadimos dos dígitos, obtenemos un resultado de dos dígitos, que envuelve la suma y el transporte de dos dígitos, como se muestra en la tabla 3.1.

a	b	Transporte	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 3.1. Suma y Transporte de dígitos en la adición binaria.

Fijémonos en las siguientes Tablas de Verdad.

p	q	$p \equiv q$	$p \nleftrightarrow q$	$p \wedge q$
V	V	V	F	V
V	F	F	V	F
F	V	F	V	F
F	F	V	F	F

Ahora bien, siguiendo nuestra discusión de la tabla 3.1, veamos que pensando el 0 y el 1 como F y V respectivamente:

$$0 \longrightarrow F$$
$$1 \longrightarrow V$$

entonces el dígito que se transporta en a + b viene dado por $a \wedge b$ y el dígito suma por $a \nleftrightarrow b$,

Es decir:

$$\text{transporte} \longrightarrow a \wedge b$$
$$\text{suma} \longrightarrow a \nleftrightarrow b$$

La adición puede, por tanto, efectuarse usando las operaciones lógicas de conjunción y no-equivalencia junto con una operación de transporte que mueve el dígito transportado sobre la posición siguiente del vector. (de derecha a izquierda).

Ejemplo 3.2

Damos el esquema de la suma de $\mathbb{V}_1$ y $\mathbb{V}_2$.

V_1 : 0 1 1 0 1 0

V_2 : 1 0 1 1 0 1

$Suma_1$: 1 1 0 1 1 1 $Sobrante_1$: 0 0 1 0 0 0

Transpórtese el sobrante una posición (010000) y súmese de - nuevo:

$Suma_1$: 1 1 0 1 1 1

(transportado)

$Sobrante_1$: 0 1 0 0 0 0

$Suma_2$: 1 0 0 1 1 1 $Sobrante_2$: 0 1 0 0 0 0

Se repite hasta que no haya sobrante alguno, transportando - siempre el sobrante en cada vez:

$Suma_2$: 1 0 0 1 1 1

$Sobrante_2$: 1 0 0 0 0 0

$Suma_3$: 0 0 0 1 1 1 $Sobrante_3$: 1 0 0 0 0 0

$Suma_3$: 0 0 0 0 1 1 1

$Sobrante_3$: 1 0 0 0 0 0 0

$Suma_4$: 1 0 0 0 1 1 1 $Sobrante_4$: 0 0 0 0 0 0 0

Esta $Suma_4$ es el resultado deseado.

Análogamente, las otras operaciones aritméticas pueden rela-- cionarse con nuestra lógica básica. Por ejemplo, la multipli cación la relacionamos, según el dígito de corrimiento o trans porte y el dígito producto, con la contradicción (conjunción- y negación) y la conjunción, respectivamente, según muestra - la Tabla 3.2.

a	b	Transporte	Producto
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1

Tabla 3.2 Suma y transporte de dígitos en la multiplicación binaria.

Transporte $\longrightarrow$ $a \wedge \sim a$

Producto $\longrightarrow$ $a \wedge b$

Vemos así que podemos tomar una sencilla notación vectorial - de muy distintas maneras, y que, en realidad, sea cual sea la interpretación que escojamos, las operaciones básicas están - relacionadas muy estrechamente. De aquí que se pueda cons- - truir un diseño, utilizando estas operaciones básicas (por -- ejemplo, negación o complementación, conjunción, y un "corrimiento"), permitiendo al usuario en potencia que interprete - los resultados de cualquier cálculo en la forma más acorde a- sus necesidades. Este es un modelo de una Máquina Lógica bastante sencilla.

CAPITULO IV

LA LOGICA MATEMATICA EN LA CIENCIA DE LA COMPUTACION.

a) Computabilidad Efectiva (Tesis de Church).

b) Algoritmos y Máquinas Computadoras.

c) Lenguajes Formales.

INTRODUCCION AL CAPITULO IV.

Al finalizar el capítulo anterior se describió una "máquina"-lógica extremadamente sencilla, donde íbamos a interpretar -- los resultados según nuestras necesidades. Ahora toca preguntarnos, ¿qué clase de problemas se pueden resolver con una máquina como la descrita?. Para expresarlo de otra forma, ¿qué es lo que se necesita pedir al problema y a la máquina para - que podamos describir el problema a la máquina, al igual que- un método para resolverlo y esperar que la máquina resuelva - el problema?.

En la parte b) discutiremos estas cuestiones y describiremos- con más detalle algunos métodos de resolución de problemas y- sus limitaciones. La Lógica que hemos desarrollado nos servirá tanto directamente para los ejemplos, como indirectamente, para base de discusión. Esta parte del capítulo se basa inicialmente en los trabajos de Turing y Markov, pues, a mi parecer, estos dos enfoques son los que ponen de manifiesto más - claramente los conceptos involucrados. Sin embargo, como las máquinas de Turing y los algoritmos de Markov son en la práctica herramientas de cálculo muy pobres, he presentado también ejemplos de algoritmos escritos en los lenguajes usados en -- computación: diagramas de flujo, un lenguaje de ensamble rudimentario, y el lenguaje BASIC.

Ya la parte c) es una continuación al estudio de los lenguaje formales en relación con los lenguajes de las computadoras y los problemas de la traducción mecánica de las lenguas- naturales.

En la parte a) damos el antecedente histórico que pone de manifiesto el ambiente y el orden en que se empezaron a tratar- los conceptos y trabajos aquí mencionados.

a) COMPUTABILIDAD EFECTIVA (Tesis de Church).

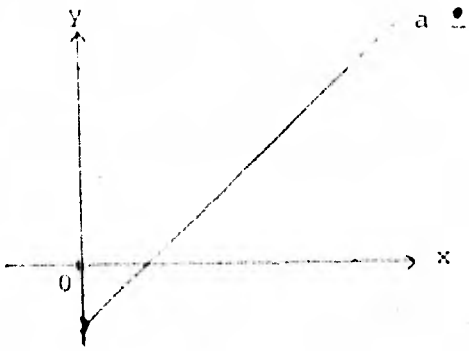
Consideremos la siguiente función real de una variable real:

$$a \dot{-} 1 = \begin{cases} a-1 & \text{si } a \geq 0 \\ 0 & \text{si } a = 0 \end{cases}$$

su dominio es $Df = \{0\} \cup \{x \in \mathbb{R} \mid x > 0\}$

su contradominio $Imf = \mathbb{R}^{+}(=\{x \in \mathbb{R} \mid x \geq 0\}) \cup (-1,0)$

y su gráfica:



Nuestro interés por esta simple función no es técnico, sino - meramente histórico. La situación en 1935 era que cierta clase exactamente definida de funciones numérico-teóricas consideradas por Alonzo Church y Stephen Kleene, durante 1932-35, llamadas "funciones λ-definibles", se encontró que tenían propiedades que sugerían fuertemente que tal clase podría abarcar a todas las funciones que pudieran ser vistas como computables bajo nuestra vaga noción intuitiva. Este resultado fue un tanto inesperado, ya que inicialmente no estaba claro si la clase contenía aún a la función particular computable $a \dot{-} 1$ descrita arriba, y una prueba en 1932 (publicada en 1935) de que así era fue hecha por S. C. Kleene como su primer trabajo de investigación matemática. Otra clase de fun--

ciones computables llamadas las "funciones generales recursivas", definidas por Gödel en 1934 basándose en una sugestión de Herbrand, tenía propiedades similares. Fue probado por Church en 1936 y Kleene (también en 1936) que las dos clases son la misma. i.e. cada función λ-definible es general recursiva y viceversa. Es evidente que toda función general recursiva (o λ-definible) es efectivamente computable.

Bajo estas circunstancias Church propuso la tesis (publicada en 1936) de que TODAS LAS FUNCIONES QUE INTUITIVAMENTE PODEMOS CONSIDERAR COMO COMPUTABLES, o con sus propias palabras - "EFECTIVAMENTE CALCULABLES", son λ-DEFINIBLES, o equivalentemente GENERALES RECURSIVAS.

Esto es una tesis más que un teorema, en tanto propone identificar un concepto intuitivo algo vago con un concepto descrito en términos matemáticos exactos, y por tanto no es susceptible de prueba. Mas Church adujo una evidencia muy fuerte, y subsecuentemente otros, en favor de la tesis.

Un poco más tarde, pero independientemente, en el trabajo de Turing de 1936-7, fue introducida otra clase exactamente definida o intuitivamente computable de funciones, las cuales llamaremos "funciones computables de Turing", y la misma proposición se hizo para esta clase; esta proposición la llamamos TESIS DE TURING. En 1937 Turing demostró en forma breve que sus funciones computables son las mismas que las funciones λ-definibles, y por tanto las mismas que las funciones generales recursivas. Así, las tesis de Turing y de Church son equivalentes. Usualmente uno se refiere a ambas como LA TESIS DE CHURCH, o en conexión con alguna de sus tres versiones que trata sobre "máquinas de Turing", como la TESIS DE CHURCH-TURING. Por el mismo tiempo, en 1936, Post publicó independientemente de Turing, una formulación más breve que fundamentalmente es la misma que la de Turing. Y en 1943 publi-

có una cuarta formulación equivalente, usando ideas de un trabajo inédito suyo de 1920-22. Aún otra formulación equivalente es proporcionada por la Teoría de Algoritmos de Markov en 1951. En los apartados que siguen de este capítulo, damos a conocer estos trabajos en una versión elemental. En el apartado b), las máquinas de Turing y los Algoritmos de Markov, y en el apartado c) los Lenguajes de Post.

b) ALGORITMOS Y MAQUINAS COMPUTADORAS.

1. Algoritmos: métodos de resolución de problemas.
2. Características y descripciones de - algoritmos.
3. Algoritmos de Markov.
4. Máquinas de Turing.
5. Los problemas del castor laborioso y la parada.
6. Computadoras digitales.
7. Lenguajes de programación.

1. ALGORITMOS: Métodos de resolución de problemas.

Comenzamos con ciertas generalizaciones que son aplicables tanto al hombre como a la máquina.

En primer lugar, si no se conoce un método para resolver un problema dado, debe entonces haber una comprensión más o menos completa del problema para que podamos llegar a una solución. De donde, supondremos en nuestro trabajo que conocemos un método para resolver un problema dado.

En segundo lugar, si se conoce un "buen" método para resolver un problema, entonces el problema no es necesario que se entienda en absoluto. Por ejemplo, un empleado puede llevar los libros de cuentas de una pequeña firma comercial sin entender las finanzas de la firma, con tal de que sepa aritmética, se le explique exactamente lo que debe hacer y haga exactamente lo que se le ha dicho. En una situación tal, debemos especificar exacta y completamente el método de solución en un lenguaje que el dispositivo que empleemos pueda interpretar.

Tercero: los problemas los podemos agrupar en conjuntos de problemas que son semejantes en algún sentido. En general, un método para resolver todos los problemas de un conjunto dado se considera superior a un método para resolver solamente un problema del conjunto.

Y por último, Cuarto: hay criterios para determinar cuán "buena" es una solución, distintos del de ver si trabaja o no, o hasta qué punto es general la aplicabilidad del método. Estos criterios envuelven cosas tales como eficiencia, elegancia, velocidad, ... pero lo más frecuente es que estén muy mal definidos.

Tratemos de definir más exactamente lo que entendemos por un método de solución de un problema.

Si se nos da un problema,

- puede ser que exista una solución (encuéntrese una x tal que $2x = 6$).
- o puede ser que no exista (encuéntrese una x tal que $0.x = 6$).
- en algunos casos, es posible que no sea nada claro que el problema tenga solución o no. Por ejemplo en áreas tales como las de las ciencias sociales es difícil, a menudo, conseguir que la gente se ponga de acuerdo en lo que constituye una solución.

Nosotros insistiremos en que cuando un problema tenga una solución bien definida, nuestro "método de solución" la encuentre; pero en caso de que no exista una solución, nuestro "método de solución" puede reconocer este hecho, pero también -- puede no reconocerlo.

El método de solución debe, desde luego ser compatible con el dispositivo usado para resolver el problema. Es decir, debe ser enunciado en un lenguaje que el dispositivo pueda comprender, y debe ser un procedimiento que el dispositivo sea capaz de ejecutar.

Ejemplo 1.1

Consideremos el problema de nuestra infancia de extraer dulces de una máquina expendedora de dulces.

el dispositivo: es la máquina expendedora de dulces.

el lenguaje: es muy sencillo. Consiste en una moneda y un empujón a la palanca de la máquina.

La máquina entonces "entiende" que una moneda como esa, seguida de un empujón a la palanca quiere decir "emite un dulce", y reacciona de acuerdo con tal orden.

Por otra parte, podríamos considerar que el dispositivo consiste en el tendero de la esquina, detrás de un mostrador que exhibe los dulces; en cuyo caso el lenguaje se torna mucho más complejo. Y la solución al problema puede describirse verbalmente a este "dispositivo" en un español ordinario.

Vemos pues, que para describir más completamente lo que entendemos por un método de solución, debemos tener alguna caracterización del dispositivo que vamos a usar. En general, podemos suponer que el dispositivo opera por pasos discretos, que tienen un comienzo y un fin bien definidos. La operación dentro de un paso puede ser continua o no; no importa para nuestros propósitos. Además, los pasos pueden efectuarse uno tras otro o paralelamente, la distinción no es importante para las partes iniciales de nuestra discusión.

Es necesario también suponer que el dispositivo puede comunicarse con el mundo exterior a través de algún lenguaje; debe decírsele qué problema se va a resolver y cómo se va a resolver, y debe dar los resultados al usuario.

Supongamos pues un dispositivo M que satisface los antedichos requerimientos generales, y un problema P.

Definición 1.1

- Un método de solución para el problema P sobre el dispositivo M es una descripción en un lenguaje comprensible para M de pasos discretos que M puede efectuar y de una ordenación de estos pasos, de modo que, dados los datos - -

apropiados, si M efectúa los pasos prescritos en el orden prescrito, tendremos una solución del problema P, - si es que tal solución existe.

- Un método de solución se llamará un semialgoritmo para- P sobre M, si la solución a P si es que existe, aparece después de la ejecución de un número finito de pasos.

- Un semialgoritmo se llamará un algoritmo si, además, - siempre que el problema no tenga solución, el dispositivo lo determina después de un número finito de pasos y se para.

En otras palabras, cuando se usa un semialgoritmo, el dispositivo intenta (casi ciegamente) resolver un problema y lo - logra después de un número finito de pasos, si el problema - es soluble. Pero cuando no existe solución alguna, un semialgoritmo puede hacer que el dispositivo usado continúe en - busca de una solución por siempre de manera infructuosa.

Un algoritmo, en cambio, resuelve también (inicialmente o al mismo tiempo) el problema asociado: ¿Tiene solución el pro-- blema dado?.

Cabe señalar que "el número finito de pasos" puede, en realidad, ser un número finito muy grande, y cada paso puede ser- bastante largo y complicado, según cual sea el dispositivo - usado.

Ejemplo 1.2

Problema: Encuéntrese una x tal que $2 + x = 5$

Algoritmo: Imprímase "3".

Ejemplo 1.3

Problema: Lados a y b como datos, encuéntrese una x tal que a + x = b.

Para detallar el algoritmo, debemos hacer algunas hipótesis específicas acerca del dispositivo que va a usarse para resolver el problema, a fin de poder presentar la solución apropiadamente.

En este ejemplo, suponemos que el dispositivo:

- puede leer los datos e imprimir la respuesta.
- es capaz de hacer comparaciones y de actuar de acuerdo con el resultado de esas comparaciones.
- tiene algún método de almacenar los datos y los resultados intermedios.
- reconoce los enteros no negativos 0, 1, 2, 3...

Ahora bien, si el dispositivo puede efectuar restas, la solución al problema es muy sencilla; pero supongamos que el dispositivo sólo puede efectuar adiciones. Debemos entonces formar a, a+1, a+2, a+3,..., y comparar cada uno de estos resultados con b. Podemos entonces tener el siguiente método de solución (para ser escrito en un lenguaje que el dispositivo pueda leer):

(1) Leánse los datos a y b.

(2) Hágase el valor de x igual a 0.

(3) Fórmese a + x.

(4) Si a + x = b, imprímase el valor de x y párese; de otra forma, continúese.

(5) Auméntese el valor de x en 1.

(6) Vuélvase al paso 3.

Nótese que éste es un semialgoritmo: si hay una x no negativa tal que a + x = b, este procedimiento la encontrará en un número finito de pasos. Pero, si no hay una tal x (por ejemplo en 4 + x = 3) entonces el método de solución repite por siempre el ciclo.

Podemos transformar esto en un algoritmo, evitando esta repetición indefinida del ciclo. Si la máquina puede percibir la relación "mayor que", entonces basta con añadir el siguiente paso 3.5 entre los pasos 3 y 4.

3.5 Si a + x es mayor que b, imprímase "no hay contestación" y párese; en caso contrario, continúese.

Si el dispositivo puede solamente comprobar la igualdad, aún podemos desarrollar un algoritmo, con tal de que a y b sean siempre enteros no negativos (para este problema);

(1) Leánse los datos a y b.

(2) Hágase el valor de x igual a 0.

(3) Fórmese a + x.

(4) Si a + x = b, imprímase el valor de x y párese; en otro caso continúese.

(5) Fórmese b + x.

(6) Si b + x = a, imprímase "no hay contestación" y párése; en otro caso, continúese.

(7) Auméntese el valor de x en una unidad.

(8) Vuélvase al paso 3.

A fin de familiarizarnos más con lo que es un algoritmo, daré

todavía 3 ejemplos más.

Ejemplo 1.4

Supongamos que tenemos un dispositivo que puede sumar, - restar, multiplicar, dividir, extraer la raíz cuadrada - (positiva) y reconocer el cero o negativo. Queremos resolver la ecuación cuadrática $ax^2 + b \cdot x + c = 0$ para x, donde a, b y c son números enteros y $a \neq 0$.

El siguiente es un algoritmo que nos la resuelve.

(1) Calcúlese $b^2 - 4ac$.

(2) Si $b^2 - 4ac$ es negativo, váyase al paso 7; en caso-contrario continúese.

(3) Calcúlese $\sqrt{b^2 - 4ac}$ y llámese d al resultado.

(4) Calcúlese (-b +d)/2a: ésta es la respuesta 1

(5) Calcúlese (-b - d)/2a: ésta es la respuesta 2.

(6) Pásese al paso 13.

(7) Multiplíquese $b^2 - 4ac$ por - 1.

(8) Calcúlese $\sqrt{-(b^2 - 4ac)}$ y llámese e al resultado.

(9) Calcúlese -b/2a.

(10) Calcúlese e/2a.

(11) Fórmese (¡no calcúlese!) -b/2a + ie/2a: esta es la respuesta 1.

(12) Fórmese -b/2a - i e/2a: ésta es la respuesta 2.

(13) Imprímanse respuesta 1 y respuesta 2, y hágase un-alto.

Nótese que en los pasos (11) y (12) decimos fórmese y no calcúlese, pues nuestro dispositivo no entiende de cálculos en -

los números complejos.

Ejemplo 1.5

Tenemos un dispositivo que puede sumar, restar y multiplicar. El siguiente algoritmo nos evalúa un determinante 3 x 3 cuyas entradas son enteros. Es decir, nos calcula:

$$\det\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = a_{11}\,a_{22}\,a_{33} + a_{12}\,a_{23}\,a_{31} + a_{13}\,a_{21}\,a_{32} - a_{13}\,a_{22}\,a_{31} - a_{11}\,a_{23}\,a_{32} - a_{12}\,a_{21}\,a_{33}$$

Sean a_{ij}, con $i, j = 1, 2, 3$, las entradas del determinante.- El cálculo de subíndices es como sigue: Si $j = 3$, cámbiese de j a 1; en otro caso, auméntese j en 1.

El cálculo de los determinantes es:

(1) Póngase SUM igual a 0.

(2) Póngase $j_1 = 1$, $j_2 = 2$, $j_3 = 3$.

(3) Calcúlese $p = a_{1j_1} \cdot a_{2j_2} \cdot a_{3j_3}$.

(4) Añádase p a SUM.

(5) Si $j_1 = 3$, váyase al paso 8; en otro caso, continúese.

(6) Calcúlese nuevos valores de j_1, j_2, j_3. (de acuerdo a como dijimos arriba).

(7) Váyase al paso 3.

(8) Hágase $j_2 = 2$, $j_3 = 1$.

(9) Calcúlese $p = a_{1j_1} \cdot a_{2j_2} \cdot a_{3j_3}$.

(10) Réstese P de SUM.

(11) Si $j_1 = 2$, imprímase SUM y hágase un alto; en otro caso continúese.

(12) Calcúlense nuevos valores de j_1, j_2, j_3.

(13) Pásese al paso 9.

Nótese que en el paso 6 no necesitamos hacer $j_1 = 3$, ya que éste tiene ese valor. Este ejemplo ilustra la especificación independiente de una subtarea o subrutina que se usa frecuentemente en la solución de un problema; en este caso el cálculo de subíndices.

Los ejemplos hasta aquí dados, son (excepto el 1.1) ejemplos numéricos. Hay, desde luego, muchos problemas no numéricos para los que se pueden encontrar algoritmos resolutivos, como enseguida se muestra.

Ejemplo 1.6

Supongamos que queremos atravesar un laberinto simple, el cual no tiene curvas cerradas (que vuelvan al punto de partida). Podemos efectuar esto, simplemente, escogiendo una dirección específica que seguiremos siempre.

1. Siempre que encontremos una rama del laberinto, escogeremos la que se encuentre más a la derecha.

2. Siempre que encontremos un punto sin salida, daremos media vuelta y continuaremos.

Para ciertos dispositivos particulares (tales como, por ejemplo, los seres humanos), se puede basar un método de solución en estas dos reglas.

He incluido este último ejemplo porque los laberintos sin cur

vas cerradas son árboles sencillos, en el sentido de la teoría de las gráficas, y uno se encuentra con que éstos aparecen frecuentemente en problemas de investigación, desde buscar un número de teléfono hasta seguir el rastro de una oscura referencia técnica. Desde luego, como personas, usamos técnicas de investigación más refinadas: uno de los mayores problemas al instruir un dispositivo mecánico en la resolución de un problema es el de caracterizar estas técnicas más refinadas de tal forma que el dispositivo pueda utilizarlas.

A continuación veremos cómo podemos clasificar los algoritmos.

2. CARACTERISTICAS Y DESCRIPCIONES DE ALGORITMOS.

Hemos estipulado que un algoritmo permite a un dispositivo resolver un problema en un número finito de pasos.

Ahora bien, podemos clasificar los algoritmos según sea la información que tenemos acerca de este número finito de pasos. La clasificación la hacemos en tres clases, tal como muestra la tabla 2.1.

CLASE I.- número de pasos fijo, o con máximo fijo.	- cíclicos (hay contador) - no cíclicos.
CLASE II.- el número máximo de pasos depende de los datos, y puede ser calculado "a priori".	
CLASE III.-el número máximo de pasos depende de los datos, pero no se puede predecir.	

TABLA 2.1. Clasificación de los Algoritmos.

- En la Clase I están los algoritmos para los que el número de pasos es fijo o tienen un máximo fijo.

Por ejemplo, si cada paso de un algoritmo para jugar al gato es una jugada, entonces hay cuando más nueve pasos.

Y al jugar al bridge hay cuando más 319 posturas en una mano y las cartas jugadas son exactamente 52. Cualquier algoritmo para jugar al bridge estaría acotado por estos números.

Los algoritmos en esta clase I pueden ser o no ser de naturaleza cíclica o iterativa (las otras dos clases de algoritmos son generalmente iterativas).

Si los algoritmos son cíclicos o iterativos, hay entonces, al menos implícitamente, un contador que va tomando en cuenta el número de pasos que han tenido lugar.

- En la clase II el número máximo de pasos está relacionado

con los datos, de tal forma que puede ser calculado "a priori".

Por ejemplo, el procesamiento de los cheques para el pago de una nómina necesita cuando más de n x k pasos, donde n es el número de empleados y k el máximo número de pasos por cheque.

En el ejemplo 1.3 de nuestra sección anterior, el semialgoritmo envuelve exactamente 4 (b - a+1) pasos, si hay solución.

- En la clase III el número de pasos está también relacionado con los datos; pero en tal forma que no podemos predecir el número de pasos involucrado.

Tales algoritmos están a menudo relacionados con investigaciones sobre conjuntos infinitos o con procesos convergentes.

Por ejemplo, si sabemos que un número entero tiene ciertas -- propiedades, podemos entonces hacer una investigación- sobre todos los enteros, para encontrarle. Pero puede que no estemos en capacidad de predecir cuánto nos llevará encontrar ese número particular.

Análogamente, en los procesos convergentes, tenemos un test - que debe satisfacer cualquier resultado aceptable, y podemos- saber que el proceso terminará por producir (en un número finito de pasos) un resultado que satisfará el test. Pero, a- pesar de ello, puede ser también que no sepamos predecir el - número de pasos necesarios para producir ese resultado satisfactorio.

Hasta aquí nuestra clasificación de algoritmos.

Obsérvese que siempre podemos transformar un semialgoritmo en un algoritmo de la primera o segunda clase, añadiendo simple-

mente un contador. Pero, generalmente, esto transforma el problema de: "Encuéntrese una solución de..., si es que existe".

en: "Encuéntrese una solución de ..., en un número de pasos menor que n, si es que existe".

Este cambio es importante, pues existe un gran número de problemas para los que hay semialgoritmos; pero para los que no hay algoritmos excepto en el sentido restringido de "soluciones en menos de n pasos".

Llegamos ahora a lo que pudiéramos llamar la realización de los algoritmos, al porqué de su estudio. Para comenzar trate mos de responder la siguientes pregunta:

¿qué problemas pueden resolverse por algoritmos?.

De una manera intuitiva, la respuesta es:

Cualquier problema para el que podamos especificar exactamente un método finito de solución.

A menudo, la forma más conveniente de especificar un algoritmo es por medio de una carta (o diagrama) de flujo.

Un diagrama de flujo consiste en una representación gráfica de la solución a un problema donde los varios pasos para resolverlo están indicados por cajas con instrucciones escritas en ellas, y el orden en que estas instrucciones han de ejecutarse se indica por flechas.

La notación que usaremos para las cartas de flujo, se muestra en la tabla 2.2.

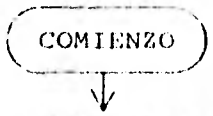
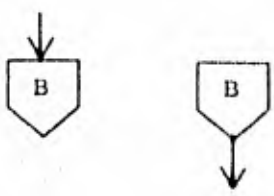
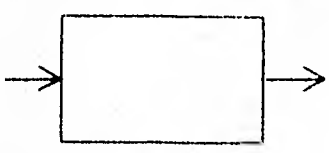
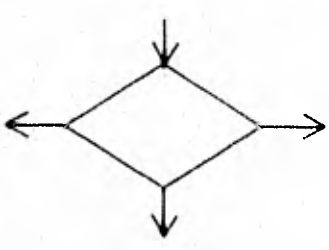
	Símbolo de comienzo del proceso
	Símbolo de final de proceso
	Símbolos conectores (es decir, los dos círculos representan el mismo punto)
	Símbolos conectores fuera de página (cuando la otra parte sigue en otra página).
	Una caja de instrucción: hágase lo que está escrito en la caja.
	Una caja de decisión: las salidas están rotuladas, y se toma la apropiada, de acuerdo con la contestación a la pregunta de la caja.

TABLA 2.2. Notación básica en una carta de flujo.

Termino de ilustrar el tema de esta sección del lenguaje de - diagramas de flujo, para tratar y describir algoritmos, con - unos ejemplos. La notación que usaré para las cartas de flujo será la de la Tabla 2.2, en todos los ejemplos.

Ejemplo 2.1

Problema: Dados a y b como datos, encuéntrese una x tal que
a + x = b.

Damos a continuación en la figura 2.1 el diagrama de flujo correspondiente al último algoritmo dado en el ejemplo 1.3 para la solución de este problema.

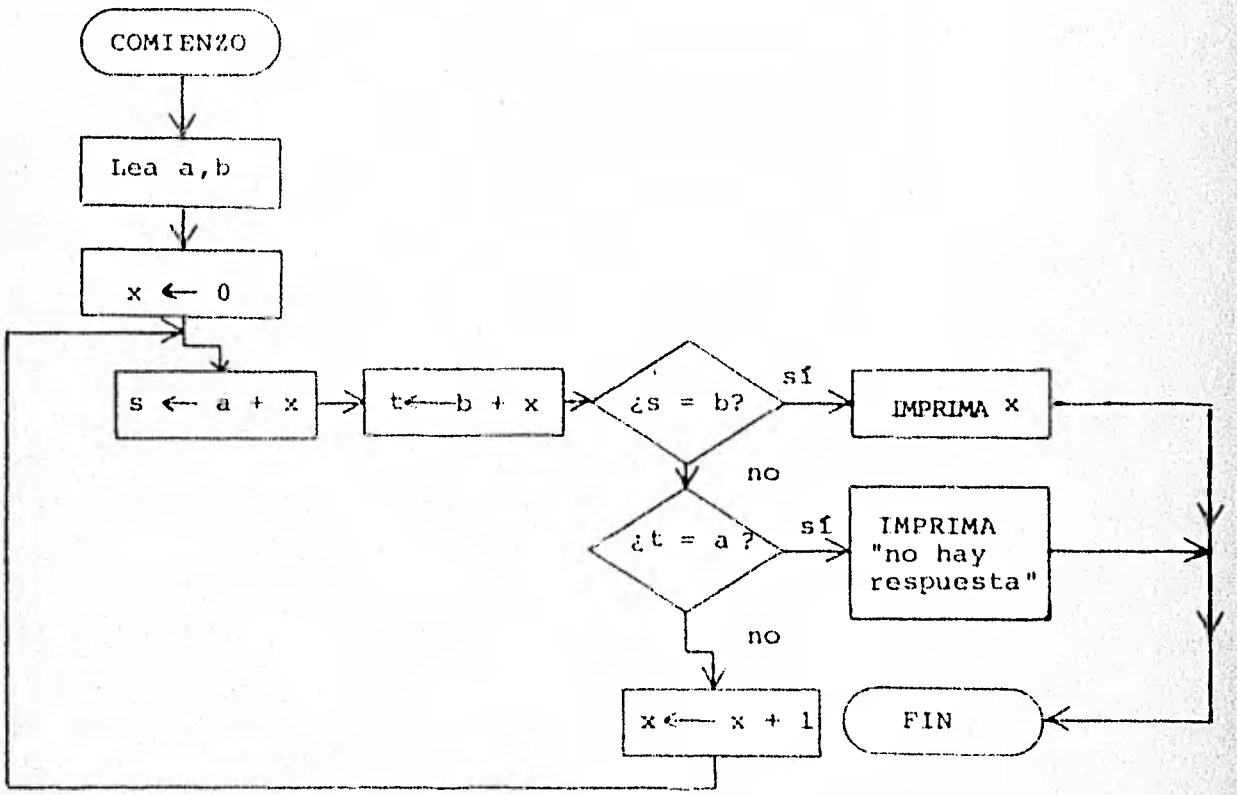


Fig. 2.1. Carta de Flujo para la resolución de a + x = b.

Ejemplo 2.2

Problema: Determínese si una fórmula lógica en notación polaca está bien formada.

Llamemos a las apariciones de los símbolos en la fórmula $s_1, \ldots, s_n$, numerados partiendo de la izquierda, y supongamos que n está dado como parte de los datos.

Una carta de flujo para el algoritmo discutido en la sección 7 del Capítulo III se da a continuación en la figura 2.2.

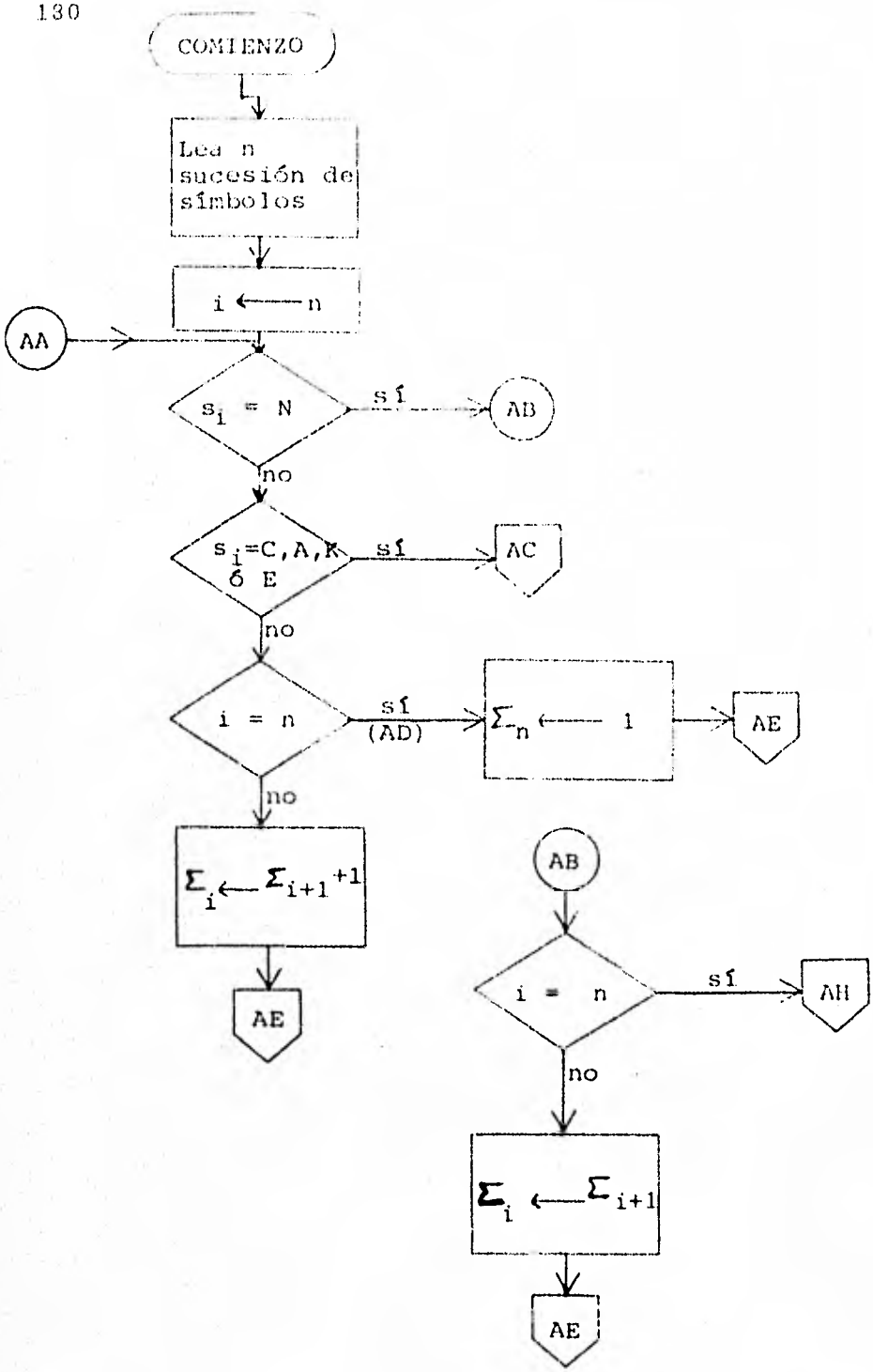


Fig. 2.2 Carta de Flujo para comprobar fórmulas bien formadas en la notación polaca.

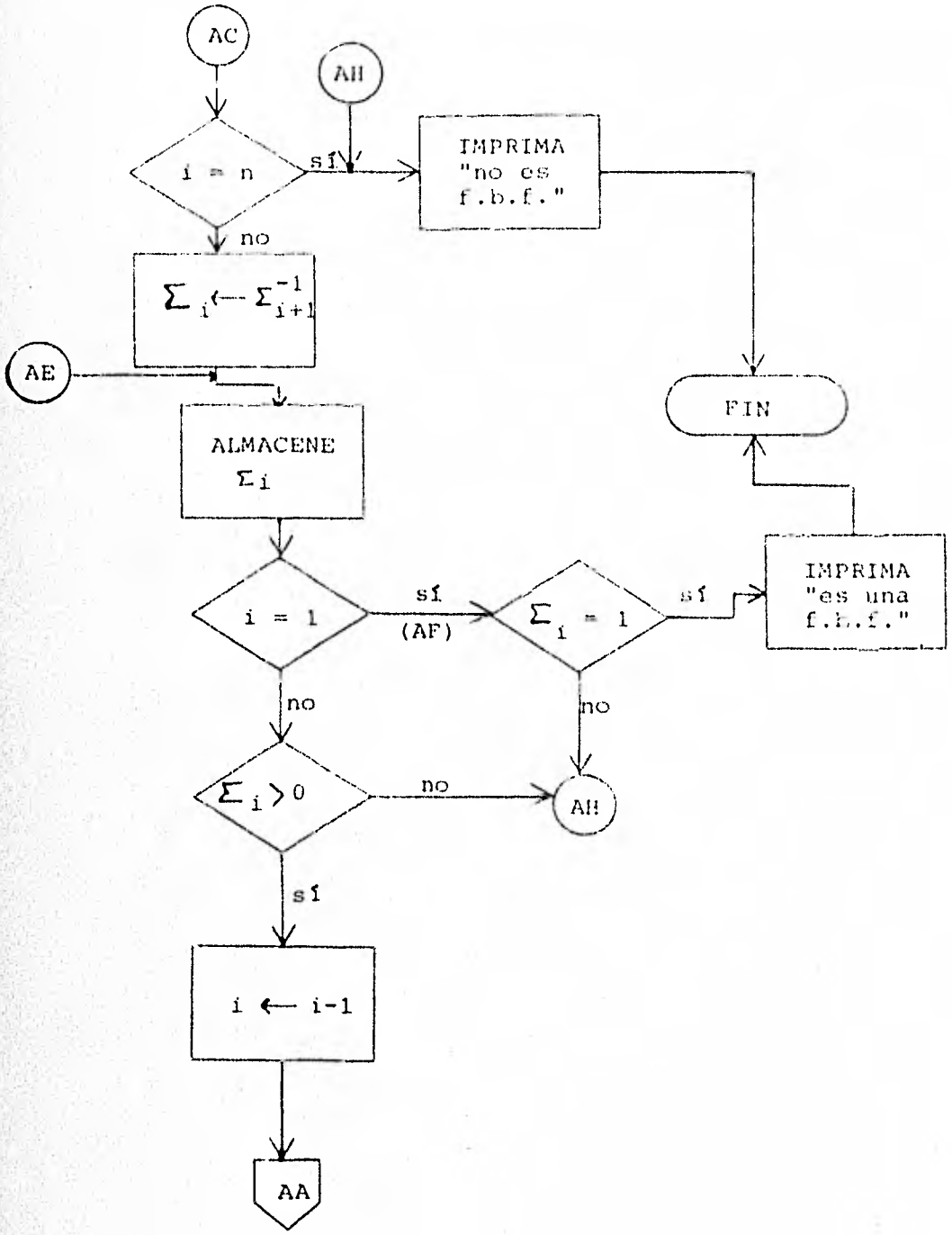


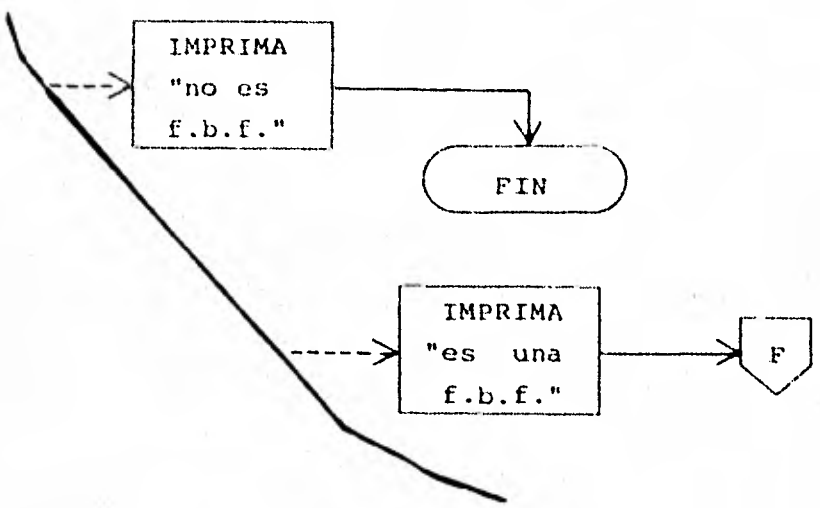
Fig. 2.2 (continuación).

Ejemplo 2.3

Problema: Desarróllese la estructura de árbol de una fórmula en notación polaca.

El Algoritmo del Arbol Polaco que para esto dimos en el Capítulo III depende de tener la sucesión Σ desarrollada, para -- comprobar que se trata de una fórmula bien formada. Supondremos entonces que ya tenemos el algoritmo para eso (como en -- realidad es, en la carta de flujo del ejemplo 2.2). Nótese - que hemos incluído explícitamente, en tal carta de flujo, la instrucción "ALMACENESE Σ_i". Esto nos asegura que podemos - disponer de la sucesión Σ para desarrollar la estructura de árbol. (El que tal instrucción de ALMACENAJE se necesite depende de cuál sea el dispositivo usado para efectuar el algoritmo).

Como el Algoritmo del Arbol Polaco es continuación del Algo-- ritmo para comprobar f.b.f. (según vimos en el Cap. III), entonces necesitamos hacer un pequeño cambio en la carta de flujo del Ejemplo 2.2. El cambio es simple: en la Fig. 2.2 (continuación) reemplazamos el (FIN) a la salida de la f.b.f. (¡ pero no a la otra salida!) por un conector, digamos [F]. Esdecir:



Entonces la carta de flujo para la estructura de árbol se continúa luego en la fig. 2.3, iniciando en la 2.2

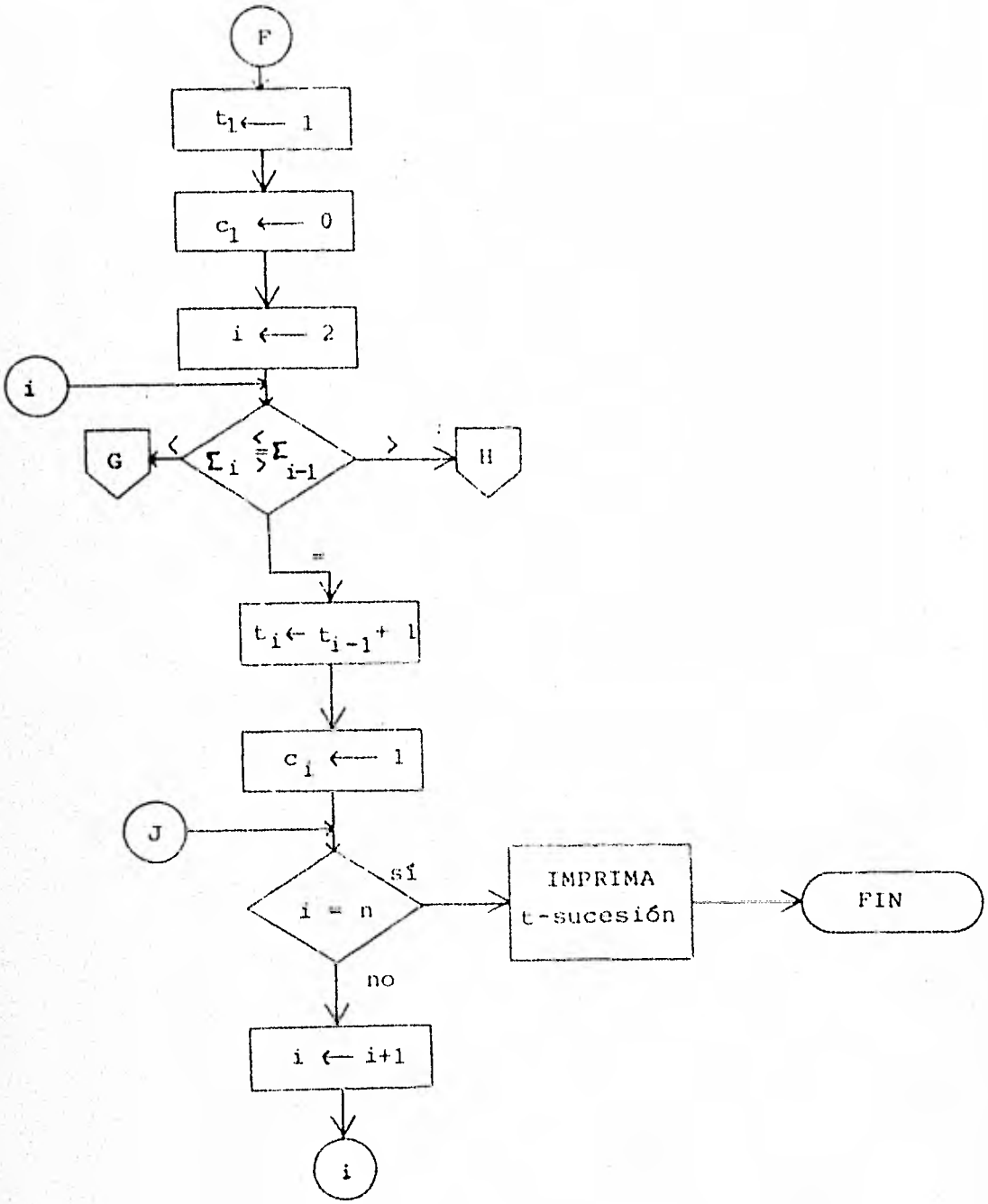


Figura 2.3 Carta de Flujo para la construcción del árbol Polaco.

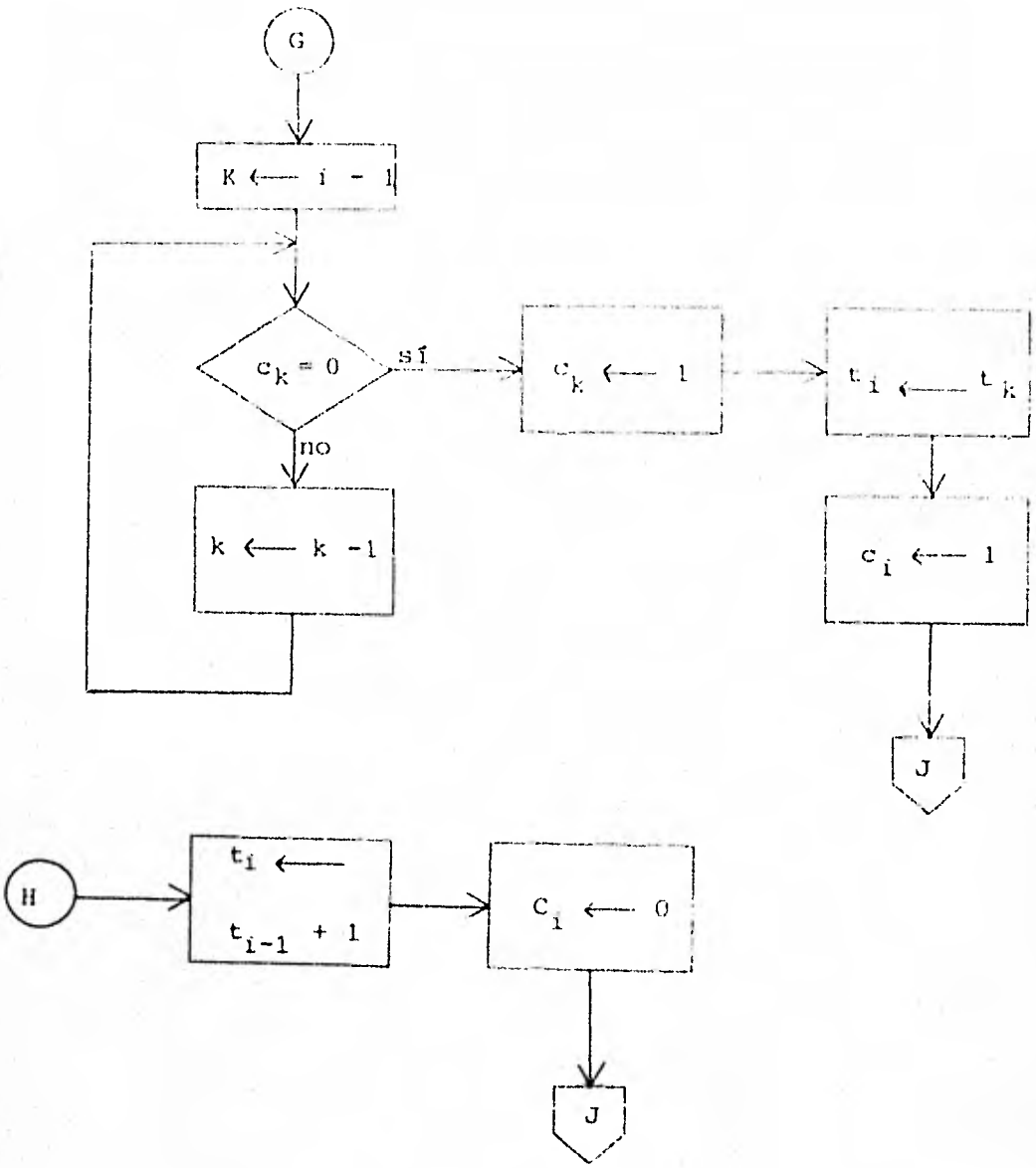


Fig. 2.3. (continuación).

Ejemplo 2.4

Problema: Resuélvanse para x y y las ecuaciones simultáneas:

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases} \quad \text{con } d,\ e,\ f \neq 0$$

Seguiré el algoritmo "por determinantes", detallado en la Fig. 2.4 por su correspondiente carta de flujo, de acuerdo a la Regla de Cramer.

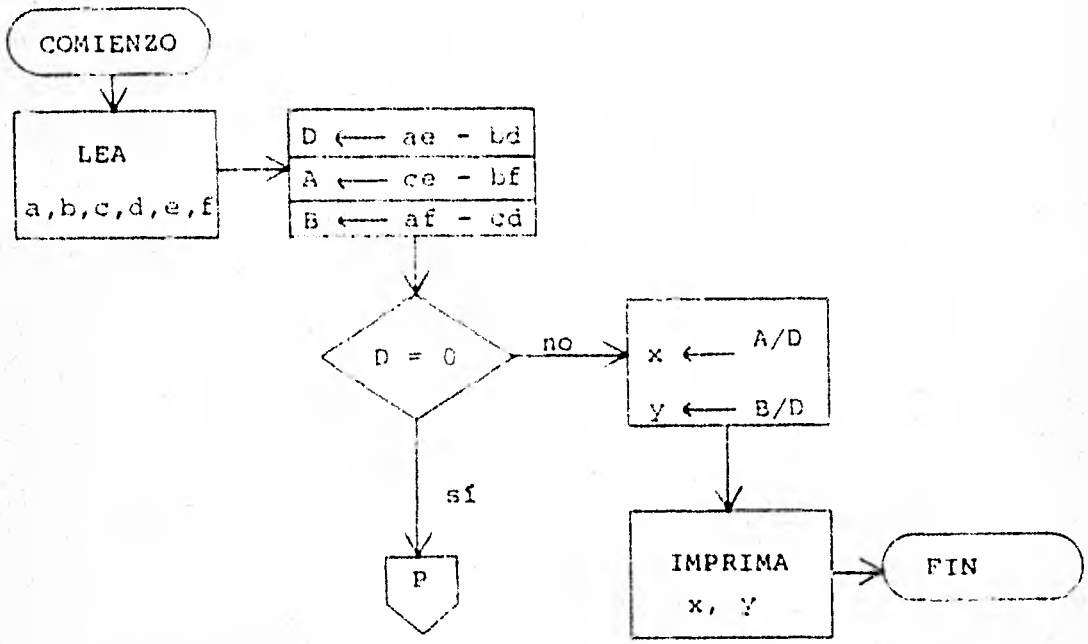


Figura 2.4. Carta de Flujo para resolver ecuaciones simultáneas dos por dos.

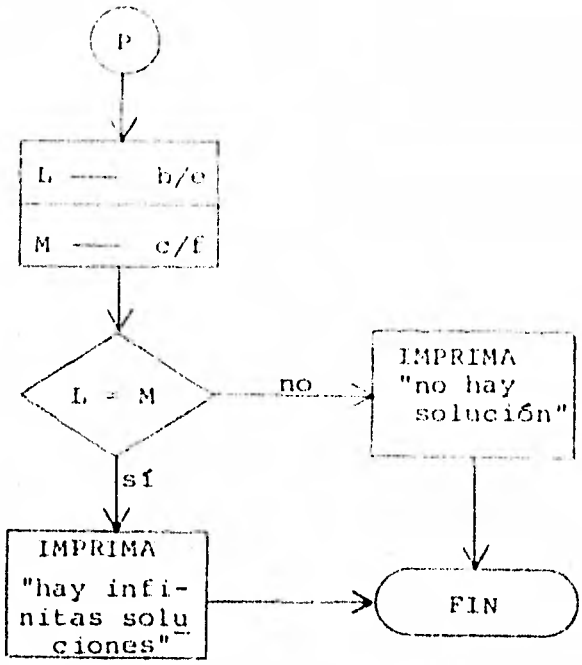
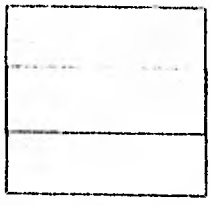
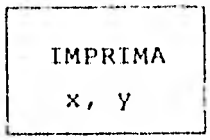


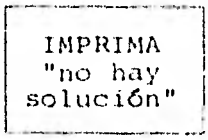
Fig. 2.4 (continuación)

OBSERVACIONES:

1. El cuadro  en realidad son tres cuadros del tipo  , sólo que por economizar espacio- cuando tracemos cartas de flujo, convenimos en resumirlos como se hizo, siempre y cuando no haya lugar a confusión respecto al orden de efectuar las operaciones.

2. Notemos que en los enunciados que se manejan en los diagramas de flujo, se utilizan dos niveles de lenguaje (Lenguaje y Metalenguaje) marcando la diferencia unas sencillas comillas " ".

   Así, el cuadro 

quiere decir que se impriman - los valores finales de x y de y . En cambio 

--

   quiere decir imprimir la frase: NO HAY SOLUCION.

Estas observaciones valen para todo este lenguaje tan descriptivo de los diagramas de flujo.

Ejemplo 2.5

Problema: Determínense todos los factores de un entero n.

Por entero vamos a entender aquí un número natural mayor que cero.

En la fig. 2.5 se detalla un algoritmo para su solución.

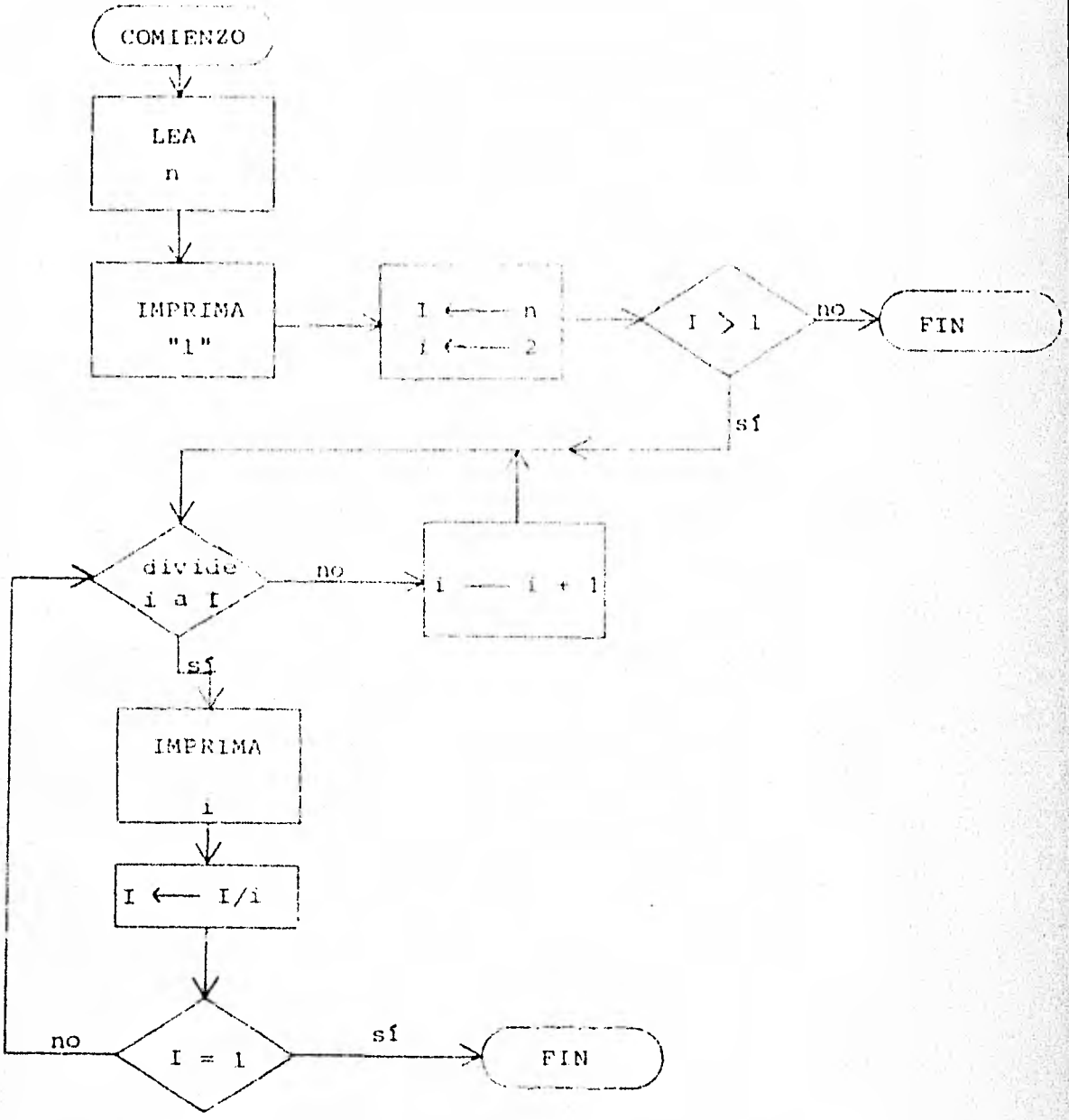


Fig. 2.5. Carta de Flujo para determinar los factores de un entero positivo.

Como último ejemplo de esta sección, veamos un típico problema de ordenamiento de datos, a menudo llamado "SORT", o sorteo.

Ejemplo 2.6.

Problema: Determínese el orden alfabético en un conjunto de n palabras.

Sean $w_1,\ldots,w_n$ las palabras del conjunto y n el número de ellas. El diagrama 2.6 nos detalla el algoritmo.

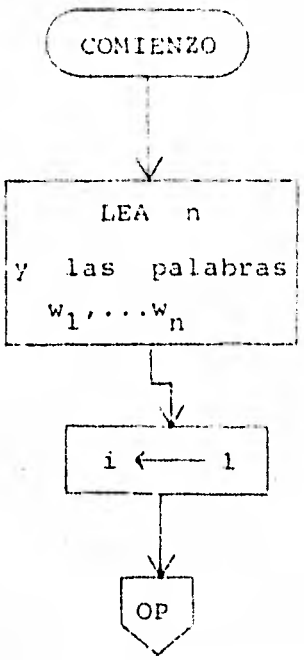


Fig. 2.6 Carta de Flujo para ordenar alfabéticamente un conjunto de n palabras.

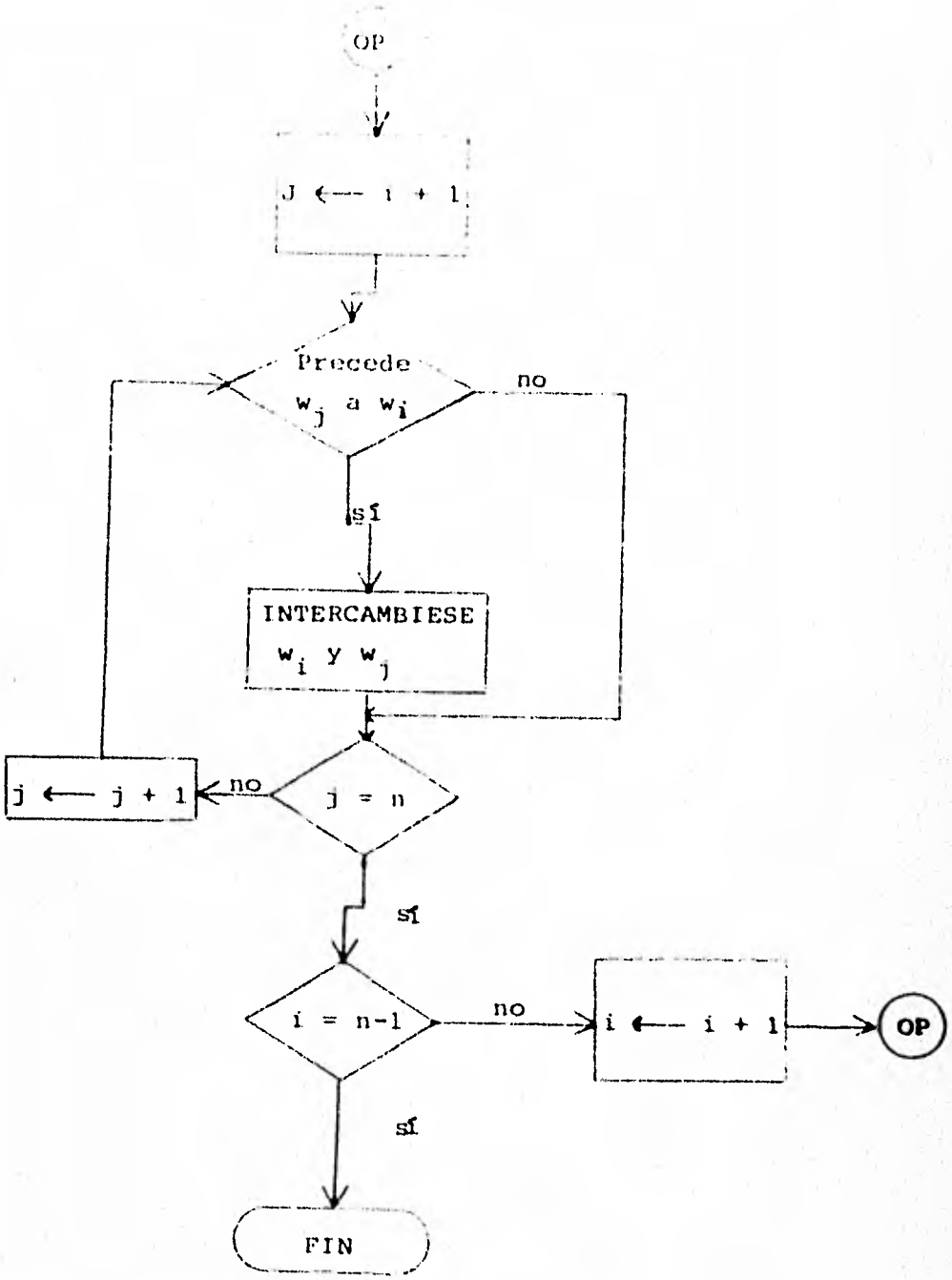


Fig. 2.6 (continuación).

Los ejemplos que he dado, ilustran el detalle de algoritmo, - incluso para tareas relativamente sencillas. Aunque el principio usado aquí puede aplicarse a la solución de cualquier - tarea, debemos darnos cuenta de que, para tareas complejas— - problemas de clasificación, de información, de mercadeo — la carta de flujo y el algoritmo para resolver el problema sobre un dispositivo particular serán más complicadas y, además, deben ser completamente explícitos. En particular, en tareas - que requieren juzgar una situación, muy intuitivamente o por un complicado análisis estadístico, el criterio de juicio debe reducirse a criterios bien definidos de "sí o no" (estado-binario), antes de que pueda formularse un algoritmo que re-- suelva el problema.

3. ALGORITMOS DE MARKOV.

En la historia de la civilización, observamos que el hombre - ha resuelto problemas basándose tan sólo en juicios intuitivos, y que, el proceso mágico de la creatividad, del descubrimiento, de la "genialidad", no consiste en el aparente "sacar formas de la nada", sino, más bien, en obtener un ordenamiento del caos, del caos de información, del caos de datos, del caos de nuestra existencia. Y obtener precisamente un ordenamiento "feliz", que funcione para nuestras necesidades, que - nos resuelva tal o cual problema y que nos sugiera un algoritmo para el futuro. Aquí reside el valor heurístico del algoritmo; más adelante, sustituye por completo al proceso de - - "buscar" cómo resolver tal o cual problema y nos lo resuelve-automáticamente. Más aún, ya en nuestro caso y por la manera en que queremos resolver los problemas (con computadora), el algoritmo se convierte en una necesidad.

Sobre el campo que nos interesa, se han propuesto muchos y -- muy diferentes enfoques para la resolución de problemas, pero

todos aquellos en los que se prescinde de la "adivinación" se han mostrado equivalentes (como dijimos en el capítulo I) en el sentido de que tratan exactamente el mismo conjunto de problemas. En ésta y la próxima sección, examinaremos dos de estos enfoques: Markov y Turing.

A. Markov, matemático ruso, en 1952 propone un procedimiento para abordar la resolución de problemas, mediante algoritmos que él llama ALGORITMOS NORMALES. (Nosotros los llamaremos ALGORITMOS DE MARKOV).

El tipo general de problema que Markov atacó es el de la - - Transformación de cadenas finitas de símbolos:

> "Dada la sucesión A, transformarla en la sucesión B de un modo mecánico."

Este problema es una abstracción de muchos de nuestros problemas más comunes.

Por ejemplo:

- El problema de sumar dos números a y b puede considerarse como el problema de transformar la sucesión - - "a + b" en una sucesión "c" que represente la suma de a y b.

- El problema de adquisición de información puede pensarse como el problema de transformar sucesiones que representen lo requerido y el conjunto de documentos, en sucesiones que representen los documentos que satisfacen lo requerido.

Al transformar una sucesión no necesitaremos, en general, operar sobre toda la sucesión (que puede ser arbitrariamente larga) de una vez, sino sólo sobre una pequeña parte contigua de ella. Supondremos que el dispositivo que tienen que utilizar

estos algoritmos es capaz de reconocer las apariciones de una subsucesión dada dentro de una sucesión dada. Estas apariciones pueden ser varias y, en realidad, puede ser que incluso se traslapen. Si deseamos distinguir una aparición particular de una subsucesión, podemos marcarla con asteriscos.

Ejemplo 3.1.

La palabra raTaTaTTaT contiene tres apariciones de la sucesión TaT, a saber:

r a * T a T * a T T a T

r a T a * T a T * T a T

r a T a T a T * T a T *

De estas apariciones, las dos primeras tienen un traslape de una letra.

Consideraremos estas apariciones como si estuvieran numeradas y nos referiremos a la aparición más a la izquierda de una sucesión A en una sucesión B como a la primera aparición de A en B, y así sucesivamente en el orden usual.

Debemos llamar la atención sobre una sucesión muy particular, a saber, la sucesión vacía, que no contiene símbolo alguno. La denotamos Λ, ésta juega un papel análogo al del conjunto vacío en el respectivo contexto. Así:

> Si una sucesión dada A contiene n símbolos, entonces se considera que la sucesión vacía Λ tiene n + 1 apariciones en A, a saber:
>
> antes del primer símbolo (la primera aparición de Λ), después --

del último símbolo (la última aparición de Λ), y entre cada dos símbolos adyacentes.

Las transformaciones de que se compone un algoritmo de Markov son las que reemplazan la primera aparición de una sucesión A especificada en la sucesión dada por otra sucesión especificada B. Un algoritmo de Markov consiste, entonces, en una sucesión de tales transformaciones, que se manipulan de una forma que más adelante daremos explícitamente.

En este momento, parece que un algoritmo de Markov podría muy bien continuar aplicando por siempre estas transformaciones o bien, puede hacer un alto porque ninguna de las transformaciones puede aplicarse a la sucesión que estamos considerando. A este respecto, deseamos ser capaces de especificar explícitamente que el proceso del algoritmo se para en un momento determinado.

Tenemos entonces los dos tipos de transformaciones que a continuación definimos:

Definición 3.1

Sean sucesiones finitas de símbolos de un conjunto finito de símbolos, llamado el alfabeto. Suponemos que el alfabeto no contiene los símbolos "$\longrightarrow$" y "$\bullet$"

- Una producción simple (de Markov) es una sucesión $A \longrightarrow B$, donde A y B son sucesiones en el alfabeto.

- Una producción conclusiva (de Markov) es una sucesión $A \longrightarrow \bullet B$, donde A y B son sucesiones en el alfabeto.

- En las producciones $A \longrightarrow B$ y $A \longrightarrow \bullet B$, el antecedente es A y el consecuente es B.

Definición 3.2.

Sea $\mathcal{A}$ un alfabeto.

Sea A $\longrightarrow$ B (ó A $\longrightarrow \bullet$B) una producción de Markov, con A y B sucesiones en el alfabeto $\mathcal{A}$.

Sea S una sucesión (o cadena) finita de símbolos en $\mathcal{A}$.

- Decimos que la producción es aplicable a S, si hay al menos una aparición de A en S. En otro caso, se dice que la producción no es aplicable a S.

NOTA: Si la producción A$\longrightarrow$ B (ó A $\longrightarrow \bullet$B) es aplicable a S y la primera aparición de A en S es P * A * Q, entonces - el resultado de aplicar la producción indicada a S es la sucesión PBQ.

Ejemplo 3.2.

Supongamos que:

Nuestro alfabeto $\mathcal{A}$ es: el alfabeto español.

La sucesión S: abactababrstc

- La producción A $\longrightarrow$ B : act $\longrightarrow$ bbb
  esta producción es aplicable a S y el resultado de aplicarla es la sucesión S': abbbababrstc.

- Al aplicar la producción ba $\longrightarrow \bullet$ one, a S obtenemos S': aonectababrstc.

- Aplicando la producción tab $\longrightarrow \Lambda$, a S , obtenemos S': abacabrstc

- Aplicando $\Lambda \longrightarrow$ x, a S obtenemos S': xabactababrstc.

- la producción abc ⟶ rst, no es aplicable a S, pues no aparece la sucesión abc en S.

Definición 3.3.

Un algoritmo de Markov es una sucesión finita $P_1,\ldots,P_n$ de producciones (de Markov) que se aplican a cadenas de símbolos en un alfabeto dado, de acuerdo con las siguientes reglas:

- Sea S una sucesión dada. (a veces llamada CADENA).

R1. Inspeccionamos la sucesión para encontrar la primera producción Pi cuyo antecedente aparece en S.

R2. a) Si no existe tal producción, la producción del algoritmo se pasa sin que S quede modificado (producción identidad).

R2. b) Si hay una producción en el algoritmo cuyo antecedente aparece en S , la primera de tales producciones se aplica a S .

R3. - Si esta producción es conclusiva, la operación del algoritmo termina sin ningún otro cambio en S .

- Si es una producción simple, hacemos una nueva inspección sobre la producción S' en que ha quedado transformada la S.

R4. Si la operación del algoritmo termina por fin -- con una sucesión S* , decimos que S* es el resultado de aplicar el algoritmo a S.

Ejemplo 3.3.

Sea nuestro alfabeto $\mathcal{A} = \{ a, b, c, d \}$.

Damos abajo el algoritmo que vamos a considerar, con las producciones numeradas para referencia.

1.- ad $\longrightarrow$ •dc

2.- ba $\longrightarrow \Lambda$

3.- a $\longrightarrow$ bc

4.- bc $\longrightarrow$ bba

5.- $\Lambda \longrightarrow$ a

- Ahora bien, sea S = dcb y apliquemos el algoritmo.

NOTACION: Voy a usar la flecha doble "$\Longrightarrow$" para indicar la transformación efectuada.

Comenzamos:

dcb $\Longrightarrow$ adcb por (5)

$\Longrightarrow$ dccb por (1)

como (1) es una producción conclusiva, el algoritmo llega a un alto con el resultado S* : dccb

- Apliquemos ahora el algoritmo a la sucesión S : dbc

dbc $\Longrightarrow$ dbba (4)

$\Longrightarrow$ db (2)

$\Longrightarrow$ adb (5)

$\Longrightarrow$ dcb (1)

De nuevo (1) es conclusivo, de forma que el resultado de aplicar el algoritmo a S: dbc es la sucesión (o cadena) S*: -- dcb

- Como tercera sucesión usemos S: bdc y apliquemos el algoritmo:

$$
\begin{aligned}
bdc &\Longrightarrow abdc && (5)\\
&\Longrightarrow bcbdc && (3)\\
&\Longrightarrow bbabdc && (4)\\
&\Longrightarrow bbdc && (2)\\
&\Longrightarrow abbdc && (5)\\
&\Longrightarrow bcbbdc && (3)\\
&\Longrightarrow bbabbdc && (4)\\
&\Longrightarrow bbbdc && (2)\\
&\Longrightarrow abbbdc && (5)\\
&\ \ \vdots
\end{aligned}
$$

la operación del algoritmo no ha terminado en este punto, y - es casi evidente que el algoritmo aplicado a bdc operará siempre, produciendo sucesiones más y más largas de la forma

b...bdc.

El algoritmo usado en este ejemplo no tiene ningún otro propósito que su uso en el ejemplo. Pero es importante observar - que si el concepto de un algoritmo de Markov ha de ser útil,-

debemos poder efectuar tareas con significado con estos algoritmos.

Ejemplo 3.4

Sea $\mathcal{A}$ un alfabeto y sea A una sucesión (cadena) fija de este alfabeto.

(Notemos que A la tenemos fija).

Deseamos transformar una sucesión arbitraria S en la sucesión AS.

Esto se consigue fácilmente con un algoritmo consistente en una sola producción, a saber:

$$\Lambda \longrightarrow\bullet\ A$$

Pero ahora supongamos que deseamos transformar S en SA.

- No podemos usar el algoritmo dado, pues aplicaciones sucesivas de éste nos darían las sucesiones AS, AAS, AAAS, --- (la primera aparición de A está al comienzo de cada palabra).

- Ni podemos escribir el algoritmo como S $\longrightarrow$ .SA, pues se necesitaría que hubiera un número infinito de producciones, una para cada S, sin ninguna primera producción, ya que se pide que la sucesión S sea arbitraria. Es decir, que el algoritmo opere para cualquier S.

En realidad, como las producciones son siempre aplicadas a la primera aparición de A en B, hay dificultades siempre que queramos operar con la segunda, tercera o última aparición.

Vencemos esta dificultad mediante el uso de símbolos marcadores especiales, que no son parte del alfabeto dado.

Hablaremos entonces de un ALGORITMO SOBRE UN ALFABETO, queriendo indicar con esto que sus producciones contienen símbolos del alfabeto y otros ciertos símbolos especificados, tales como símbolos marcadores. Por el uso de estos, podemos marcar cualquier punto particular en una sucesión y, luego, operar en ese punto.

Esto nos resuelve el problema de transformar S en SA, como se detalla en el siguiente ejemplo.

Ejemplo 3.5

Sea $\mathcal{A}$ el alfabeto dado en el ejemplo 3.4,

A la misma sucesión fija.

Sea α un marcador ($\alpha \notin \mathcal{A}$)

Si S es una sucesión en $\mathcal{A}$, entonces el resultado de aplicar el siguiente algoritmo a S es la sucesión **SA**.

1.- $\alpha\, \tau \longrightarrow \tau\, \alpha \quad (\tau \in \mathcal{A})$

2.- $\alpha \longrightarrow \bullet\, A$

3.- $\Lambda \longrightarrow \alpha$

- Como S no contiene inicialmente a α (pues $\alpha \notin \mathcal{A}$), la tercera producción se usa para obtener αS.
- La primera producción se usa entonces para mover α a la posición detrás de los símbolos de S.
- Si S contiene n ocurrencias de símbolos, entonces después de n pasos obtenemos la sucesión $S\alpha$.

- Al llegar al n-ésimo paso, la primera producción ya no se aplica, y la segunda produce SA; como esta producción es conclusiva, la sucesión SA es entonces el resultado.

OBSERVACIONES:

- En el ejemplo anterior introducimos una nueva notación, a saber: en la primera producción hemos usado la variable ζ que recorre todos los símbolos de $\mathcal{A}$. Luego la primera línea no es realmente una producción, sino más bien un ESQUEMA DE PRODUCCION, que denota todas las producciones que pueden obtenerse, sustituyendo símbolos de $\mathcal{A}$ en lugar de ζ.

- A causa de la forma en que se usan los algoritmos de Markov, el orden en que se escriben las producciones es vital.

  Por ejemplo:

  Si las primeras dos líneas del esquema de algoritmos en el ejemplo 3.5 se intercambiaran, el resultado sería - - transformar S en AS en lugar de en SA, y las producciones representadas por $\alpha\zeta \longrightarrow \zeta\alpha$ nunca se usarían.

- Observando cada línea como un esquema del esquema de algoritmos propuesto, o sea, observando en sí los propios esquemas de producción (las líneas) vemos que representan cada uno secciones del algoritmo en que el orden de cada una de las producciones individuales no es crítico, ya que, a lo más, una de las producciones representadas es la que se aplica y todas ellas hacen la misma cosa en contextos diferentes.

El tema de los algoritmos de Markov va mucho más lejos que lo expuesto en esta sección. Lo que aquí se presenta no es más

que una pequeña introducción al tema, pero suficiente para - nuestros propósitos.

El trabajo de Markov inaugura una etapa de "formalidad" en el desarrollo de la teoría de los algoritmos y, como veremos por último, facilita la resolución de problemas al tratarlos de- una manera formal y explícita, en particular:

> "Cualquier tarea que puede realizarse por el uso de algoritmos, en el sentido más amplio (de la - sección 1), puede también realizarse por el uso- de algoritmos de Markov."

Esto resulta muy interesante y favorecedor para dichos algo-- ritmos, que muestra la capacidad nada restringida de los mismos.

Concluimos esta sección con varios ejemplos de tareas que pue den realizarse fácilmente con algoritmos de Markov.

En los ejemplos, el alfabeto $\mathcal{A}$ no se especifica, pero se sabe que no contiene ninguno de los marcadores ni símbolos espe ciales que explícitamente se enuncian.

Ejemplo 3.6.

> Este algoritmo transforma toda sucesión en la sucesión - vacía.
>
> $$\zeta \longrightarrow \Lambda \quad (\zeta \in \mathcal{A})$$
>
> En operación, este algoritmo va quitando las letras de - una sucesión, comenzando con las apariciones de la iz- - quierda, claro, siguiendo los pasos de lo que es un Algo ritmo de Markov.

Ejemplo 3.7.

> Este algoritmo deja sin cambio alguno a la sucesión va--

cía, pero quita la primera letra de cualquier sucesión - no vacía y luego se detiene.

Marcador: α.

$$\alpha\, \zeta \longrightarrow \bullet \Lambda \quad (\zeta \in \mathcal{A})$$

$$\alpha \longrightarrow \bullet\, \Lambda$$

$$\Lambda \longrightarrow \alpha$$

Ejemplo 3.8

A menudo, es útil conocer el número de símbolos que tiene una sucesión.

Realizamos sencillamente esta tarea en una notación de - marcas, reemplazando cada símbolo por una marca.

Símbolo especial: 1

$$\zeta \longrightarrow 1 \quad (\zeta \in \mathcal{A})$$

Como "1" no es un elemento de $\mathcal{A}$, la operación cesa - - cuando todos los símbolos han sido transformados.

Ejemplo 3.9.

En casi todo problema llega un momento en que debe tomar se una decisión que depende de cuáles sean los resulta-- dos del cálculo efectuado hasta ese punto.

Presentamos un algoritmo de Markov para hacer una deci-- sión de tal tipo.

- Se examina una sucesión arbitraria en el alfabeto dado - para determinar si es una sucesión especificada A.

+ Si lo es, toda la sucesión se reemplaza por la sucesión-

B; en otro caso, toda la sucesión se reemplaza por la - sucesión C.

Marcador: α .

$\zeta \alpha \longrightarrow \alpha \zeta$ ( )

$\alpha \zeta \longrightarrow \alpha$ ( )

$\alpha \longrightarrow \bullet C$

$A \zeta \longrightarrow \alpha$ ( )

$\zeta A \longrightarrow \alpha$ ( )

$A \longrightarrow \bullet B$

$\Lambda \longrightarrow \alpha$

- Si la sucesión dada, digamos P, no contiene una ocu- - rrencia de la sucesión A, la última producción introduce un α y entonces los esquemas de producción segundo y tercero borran P y lo reemplazan por C.
- Si P contiene una ocurrencia de A, pero no es A, el esquema de producción cuarto o el quinto son los que entran en juego para introducir α ; el primer esquema -- mueve la α al extremo izquierdo de P y, entonces, el- segundo y el tercero operan como antes.
- Finalmente, si la sucesión P es realmente A, la producción sexta se aplica, y P se transforma en B.

Nótese que las producciones se refieren directamente a - la sucesión A, que puede ser verdaderamente larga. Como A se conoce a priori, esto es admisible: podríamos reem-

plazar siempre esa referencia por una búsqueda de A le-- tra por letra.

Ejemplo 3.10.

A veces, desea uno descartar una porción de una sucesión de símbolos como se descartan los datos luego de haber - calculado la respuesta.

El siguiente algoritmo descarta todo lo que se encuentre a la izquierda del símbolo especial α.

$$\zeta\alpha \longrightarrow \alpha \quad (\zeta \in A)$$

$$\alpha \longrightarrow \bullet\Lambda$$

Ejemplo 3.11.

Otro procedimiento que es muy común es el de doblar o duplicar una sucesión. A menudo, deseamos efectuar transformaciones que destruyen una sucesión, pero que sólo -- son de naturaleza tentativa: en un cierto momento, podemos decidir que las transformaciones eran equivocadas y desear comenzar de nuevo. Debemos, pues, poder salvar - una copia (backoff) de la sucesión original a la que podamos volver.

Dada una sucesión P, los siguientes algoritmos producen la sucesión PP antes de que lleguen a un alto.

Marcadores: α, β, η.

1.- $\zeta\,\eta\beta \longrightarrow \eta\beta\zeta$ $\quad(\zeta, \eta \in \mathcal{A})$

2.- $\alpha\zeta \longrightarrow \zeta\beta\zeta\alpha$ $\quad(\zeta \in \mathcal{A})$

3.- $\beta \longrightarrow \eta$

4.- $\eta \longrightarrow \Lambda$

5.- $\alpha \longrightarrow \bullet\Lambda$

6.- $\Lambda \longrightarrow \alpha$

Para observar cómo opera este esquema de algoritmos, apliquémoslo a la sucesión, digamos P: 01 con $\mathcal{A} = \{0, 1, 2\}$

P

01 $\Longrightarrow$ α 01 (6)

$\Longrightarrow$ 0 β 0 α 1 (2)

$\Longrightarrow$ 0 β 01 β 1 α (2)

$\Longrightarrow$ 0 β 1 β 01 α (1)

$\Longrightarrow$ 0 δ_1 1 ε 01 α (3)

$\Longrightarrow$ 0 δ_1 1 δ_1 01 α (3)

$\Longrightarrow$ 01 δ_1 01 α (4)

$\Longrightarrow$ 0101 α (4)

$\Longrightarrow$ $\underbrace{01}_{P}\underbrace{01}_{P}$ (5)

Ejemplo 3.12.

Como ejemplo final, damos un algoritmo de proyección.

Estamos tratando con sucesiones de la forma $A_1 \alpha A_2 \alpha \ldots \alpha A_n$, donde α es un símbolo especial.

La tarea es aislar de esta sucesión la subsucesión A_i, donde $i \leq n$ y ambos son conocidos.

Se consigue esto, borrando $A_1, A_2, \ldots, A_{i-1}$, pasando so--

bre A_i y luego, borrando la porción restante de la sucesión.

Marcadores: $\beta_1, \ldots, \beta_i, \beta'$

$$\beta_1 \gamma \longrightarrow \beta_1 \qquad (\gamma \in \sigma - \alpha)$$
$$\beta_1 \alpha \longrightarrow \beta_2$$
$$\beta_2 \gamma \longrightarrow \beta_2 \qquad (\gamma \in \sigma - \alpha)$$
$$\beta_2 \alpha \longrightarrow \beta_3$$
$$\vdots$$
$$\beta_{i-1} \gamma \longrightarrow \beta_{i-1} \qquad (\gamma \in \sigma - \alpha)$$
$$\beta_{i-1} \alpha \longrightarrow \beta_i$$
$$\beta_i \gamma \longrightarrow \gamma \beta_i \qquad (\gamma \in \sigma - \alpha)$$
$$\beta_i \alpha \longrightarrow \beta'$$
$$\beta' \gamma \longrightarrow \beta'$$
$$\beta' \alpha \longrightarrow \beta'$$
$$\beta' \longrightarrow \bullet \Lambda$$
$$\Lambda \longrightarrow \beta_1$$

4. MAQUINAS DE TURING.

Pasamos ahora de nuevo a un enfoque más clásico del problema-de resolución de problemas.

En 1936, Alan Turing, matemático inglés, escribió un artículo en el que desarrollaba un método mecánico para la solución de problemas que, desde entonces, se ha conocido como una máquina de Turing.

Varios modelos, diferentes pero equivalentes, de máquinas de Turing se han propuesto desde entonces. Discutiremos solamente uno de ellos.

Daremos primero una definición formal y luego una descripción intuitiva de la operación de este tipo de máquinas Turing y varios ejemplos de su uso.

Definición 4.1.

Sean los siguientes conjuntos finitos con los nombres respectivos que se indican:

$Q = \{ q_0, q_1, \ldots, q_n \}$ conjunto de estados,

$S = \{ s_0, s_1, \ldots, s_m \}$ conjunto de símbolos,

y

$\mathcal{M} = \{ R, L \}$ conjunto de movimientos.

- Una expresión es una sucesión finita de signos, todos los cuales, excepto uno, son símbolos (es decir, elementos de S), siendo el signo excepcional un estado (es decir, elemento de Q).
- Un quíntuple es una sucesión de cinco elementos (q_i, s_j, s_k, m, q_l) en el que q_i y q_l son estados, s_j y s_k símbolos, y m un movimiento.
- Una Máquina de Turing es un conjunto finito de quíntuples, entre los que no hay dos que tengan el mismo par de elementos iniciales q_i y s_j.

Una máquina de Turing transforma expresiones en otras expresiones, de acuerdo con las siguientes reglas:

Supongamos que P y Q son sucesiones arbitrarias de símbo

los (en S), posiblemente vacías, y sea s' ∉ S.

Consideremos a s_0 como un símbolo especial, al que llamaremos un blanco, y al que denotaremos usualmente por □

CASO 1. Quíntuples de la forma $q_i\ s_j\ s_k\ Pq_l$.

- Las expresiones de la forma $Pq_i s_j s'Q$ se transforman en expresiones $Ps_k q_l s'Q$.
- Expresiones de la forma $Pq_i s_j$ se transforman en expresiones $Ps_k q_l \square$,
- Otras expresiones no se transforman.

CASO 2. Quíntuples de la forma $q_i\ s_j\ s_k\ Lq_l$

- Expresiones $Ps'q_i s_j Q$ se transforman en expresiones $Pq_l s' s_k Q$.
- Expresiones $q_i s_j Q$ se transforman en expresiones $q_l \square s_k Q$.
- Otras expresiones no se transforman

Esto es la definición y las Reglas de Transformación de una Máquina de Turing. Veamos ahora un poco más profundamente.

Cuando una Máquina de Turing se aplica a una expresión, se permite que opere hasta que alcanza un par de símbolos de estado $q_i s_j$ para los que no está definido ningún quíntuple. El resultado, en ese momento, puede considerarse como el resultado de aplicar la Máquina de Turing a la expresión inicial, aunque, más comúnmente, se considera como resultado a la sucesión de símbolos en tal momento, es decir, se ignora el estado final.

Veamos el siguiente ejemplo de una Máquina de Turing.

Ejemplo 4.1

	q_o	□	□	L	q_4		q_1	c	c	R	q_o		q_4	d	d	L	q_4
3o.→	q_o	a	a	R	q_1	4o.→	q_1	d	c	L	q_2		q_5	□	b	R	q_6
1o.→	q_o	b	b	R	q_o	5o.→	q_2	a	d	L	q_7		q_5	a	a	L	q_4
2o.→	q_o	c	c	R	q_o		q_3	a	a	L	q_5		q_5	b	b	L	q_4
	q_o	d	d	R	q_o		q_4	□	□	R	q_7		q_5	c	c	L	q_4
	q_1	□	□	L	q_3		q_4	a	a	L	q_4		q_5	d	d	L	q_4
	q_1	a	a	R	q_1		q_4	b	b	L	q_4		q_6	a	c	L	q_6
	q_1	b	b	R	q_o		q_4	c	c	L	q_4		q_6	b	b	L	q_7

Esta máquina de Turing opera sobre el conjunto de símbolos $\{□, a, b, c, d\}$ (o cualquier conjunto que contenga a éste como subconjunto).

Notemos, en esta Máquina Turing, que:

- Hay referencias entre los quíntuples a un estado q_7, pero ninguno de los quíntuples comienza con q_7. Luego q_7 es, en realidad, un estado de "parada".

- Notemos también que hay ciertos pares de símbolos de estado, por ejemplo q_2b, con los cuales no comienzan ninguno de los quíntuples que aparecen en la lista. Luego, aunque q_2 no es siempre un estado de parada, la máquina parará si, cuando el estado es q_2, se encuentra el símbolo b.

Para ilustrar la operación de esta máquina, consideremos la expresión $q_o$bcadc.

La máquina transformará esto, sucesivamente, en:

1o. $\longrightarrow$ bq_0cadc

2o. $\longrightarrow$ bcq_0adc

3o. $\longrightarrow$ $bcaq_1dc$

4o. $\longrightarrow$ bcq_2acc

5o. $\longrightarrow$ bq_7cdcc

y en este momento se parará.

Así pues, la máquina ha transformado la sucesión de símbolos- bcadc en la sucesión bcdcc. En el cuadro de quíntuples que - constituyen la máquina Turing, se señalan con flechas los - - quíntuples que intervinieron en cada paso de la transforma- - ción.

Esto corresponde a la producción de Markov.

$$ad \longrightarrow \bullet dc$$

Evidentemente, en este caso resulta mucho más simple la tarea siguiendo el método de Markov que el de Turing. Una conven-- ción que es útil al trabajar con máquinas Turing es la de una representación tabular, de la siguiente manera:

El quíntuple $q_i\ s_j\ s_k\ m\ q_l$ está representado por la entrada - de la tabla $s_k\ m\ q_l$ con coordenadas q_i y s_j.

En esta forma, la máquina Turing de nuestro ejemplo 4.1 apare ce en la tabla 4.1.

	□	a	b	c	d
q_0	□ Lq_4	aRq_1	bRq_0	cRq_0	dRq_0
q_1	□ Lq_3	aRq_1	bRq_0	cRq_0	cLq_2
q_2		dLq_7			
q_3		aLq_5			
q_4	□ Rq_7	aLq_4	bLq_4	cLq_4	dLq_4
q_5	bRq_6	aLq_4	bLq_4	cLq_4	dLq_4
q_6		cLq_6	bLq_7		

Tabla 4.1 Máquina Turing del ejemplo 4.1

Hemos visto ya lo que es formalmente una máquina de Turing y un ejemplo concreto con una aplicación de transformación de sucesiones. Vimos también la opción de poder representar en forma de tabla una máquina de Turing. Veamos ahora lo que intuitivamente podemos entender de estas máquinas.

INTERPRETACION INTUITIVA DE UNA MAQUINA DE TURING.

Consideremos una máquina consistente en una caja negra con una cabeza lectora e inscriptora diseñada para examinar una cinta que está dividida en cuadrados. (véase la fig. 4.1). La cinta puede considerarse o infinita o arbitrariamente extensible en ambas direcciones. Dentro de la caja negra, el dispositivo puede tomar una entre un cierto número de configuraciones o estados internos. Estos proveen a la máquina de una cierta cantidad de memoria limitada. En operación, la cabeza lecto-

ra-inscriptora examina un cuadrado de la cinta. Dependiendo- de cuál sea su estado interno y el símbolo que encuentre en - la cinta, imprime otro símbolo sobre ese cuadrado, se mueve - un cuadrado a la derecha o a la izquierda, y cambia su estado.

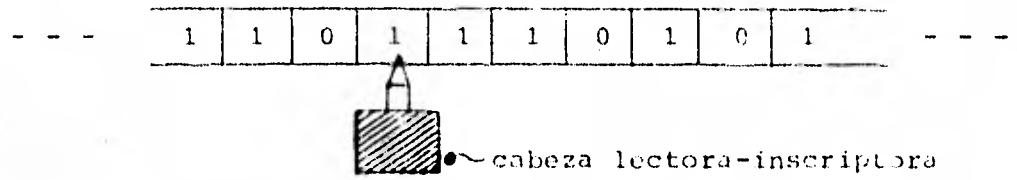


Fig. 4.1 Máquina de Turing básica.

La hipótesis formal de que una expresión es finita en longi-- tud se corresponde con la hipótesis informal de que en todo - momento la cinta está en blanco, excepto por un número finito de símbolos en ella inscritos.

Con esta misma interpretación de la máquina, veamos los res-- tantes elementos de una máquina (contador, sumador, reconocedor de números) mediante ejemplos.

Ejemplo 4.2. UN CONTADOR.

Suponemos que sobre la cinta hay una sucesión de ceros y unos acotados a la izquierda y a la derecha por el símbolo "$". Se desea contar el número de unos en esta sucesión e imprimir a la izquierda del "$" que está más a la izquierda el número decimal que representa este número de unos.

Se desea también dejar la sucesión original sin ningún - cambio, suponiendo que tenemos un símbolo "a" que no aparece y que, por lo tanto, podemos usar como marcador.

Si suponemos que inicialmente la máquina de turing está-

en el estado q_0 y está examinando el símbolo "$" situado más a la izquierda, entonces, la máquina que se muestra en la Tabla 4.2 efectúa la tarea deseada.

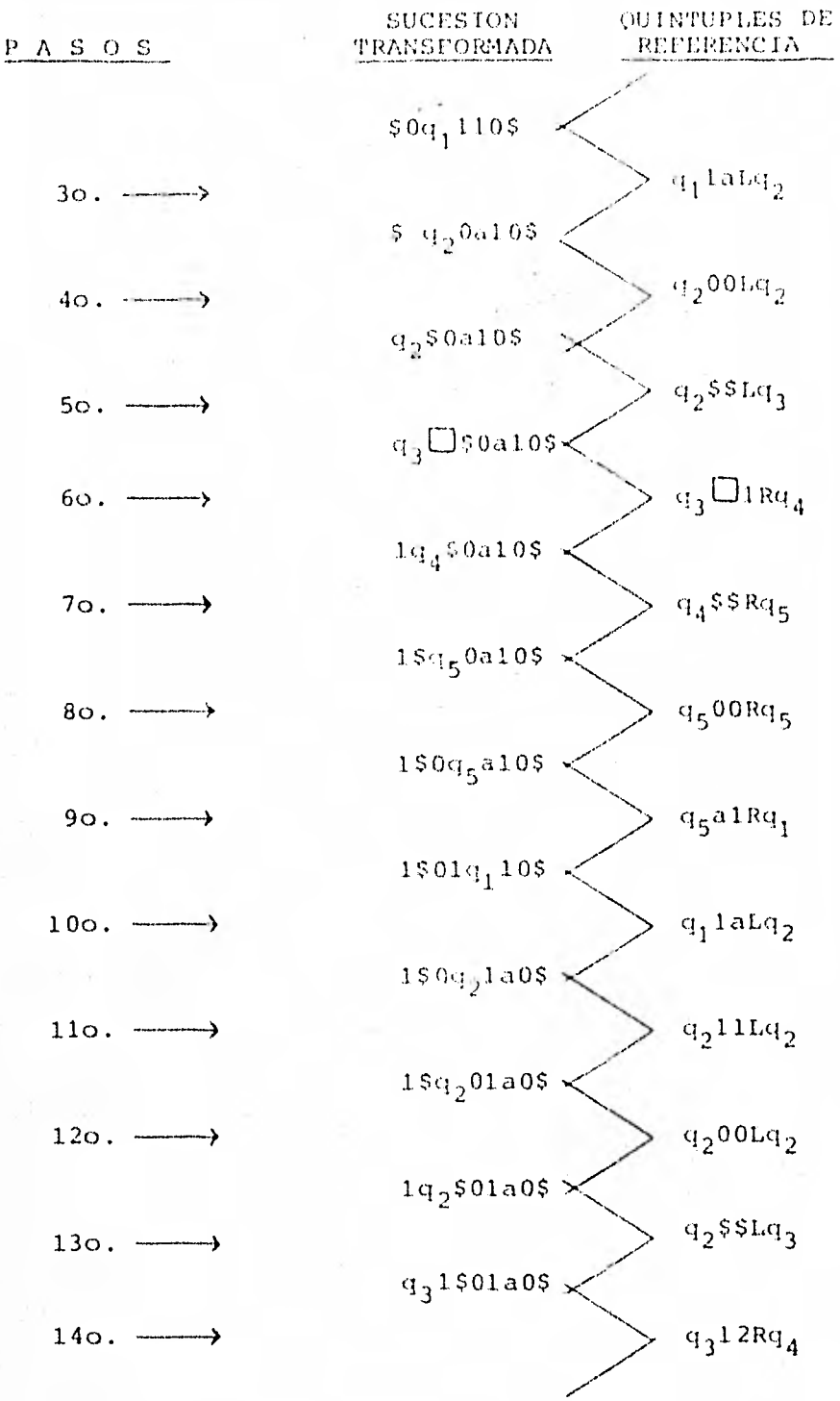
	$	0	1	2	3	4	5	6	7	8	9	a	□
q_0	Rq_1												
q_1	Rq_6	0Rq$_1$	aLq_2										
q_2	Lq_3	0Lq$_2$	1Lq$_2$										
q_3		1Rq$_4$	2Rq$_4$	3Rq$_4$	4Rq$_4$	5Rq$_4$	6Rq$_4$	7Rq$_4$	8Rq$_4$	9Rq$_4$	0Lq$_3$		1Rq$_4$
q_4	Rq_5	0Rq$_4$											
q_5		0Rq$_5$	1Rq$_5$									1Rq$_1$	

Tabla 4.2 Un contador.

Esta máquina efectúa la tarea buscando a la derecha para encontrar un "1" que reemplaza por "a". Vuelve entonces a la izquierda y añade uno a la cuenta que está formando. Después de esto, la máquina localiza la "a", la reemplaza por un "1" y prosigue la búsqueda. La localización del "$" situado más a la derecha, para el proceso.

Para verla funcionar, apliquémosla a la sucesión 0110.

PASOS	SUCESION TRANSFORMADA	QUINTUPLES DE REFERENCIA
	$q_0$$0110$	
1o. ⟶		$q_0$$$Rq$_1$
	$$q_1$$0110$	
2o. ⟶		$q_1$00Rq$_1$


PASOS
SUCESION TRANSFORMADA
QUINTUPLES DE REFERENCIA
$0q1110$
3o.
q11aLq2
$ q20a10$
4o.
q200Lq2
q2$0a10$
5o.
q2$$Lq3
q3□$0a10$
6o.
q3□1Rq4
1q4$0a10$
7o.
q4$$Rq5
1$q50a10$
8o.
q500Rq5
1$0q5a10$
9o.
q5a1Rq1
1$01q110$
10o.
q11aLq2
1$0q21a0$
11o.
q211Lq2
1$q201a0$
12o.
q200Lq2
1q2$01a0$
13o.
q2$$Lq3
q31$01a0$
14o.
q312Rq4

P A S O S	SUCESION TRANSFORMADA	QUINTUPLOS DE REFERENCIA
	$2\$q_4\$01a0\$$	
15o. ⟶		$q_4\$\$Rq_5$
	$2\$q_501a0\$$	
16o. ⟶		q_500Rq_5
	$2\$0q_51a0\$$	
17o. ⟶		q_511Rq_5
	$2\$01q_5a0\$$	
18o. ⟶		q_5a1Rq_1
	$2\$011q_10\$$	
19o. ⟶		q_100Rq_1
	$2\$0110q_1\$$	
20o. ⟶		$q_1\$\$Rq_6$
	$2\$0110\$9_6$ □	

Se para el proceso.

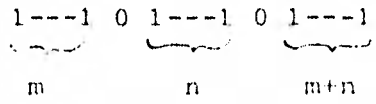
EJEMPLO 4.5 UN SUMADOR.

Nuestro conjunto de símbolos consta ahora de exactamente dos-símbolos, un "0" y un "1".

Deseamos transformar la sucesión:

$$\underbrace{1\text{---}1}_{m} \quad 0 \quad \underbrace{1\text{---}1}_{n}$$

en la sucesión:



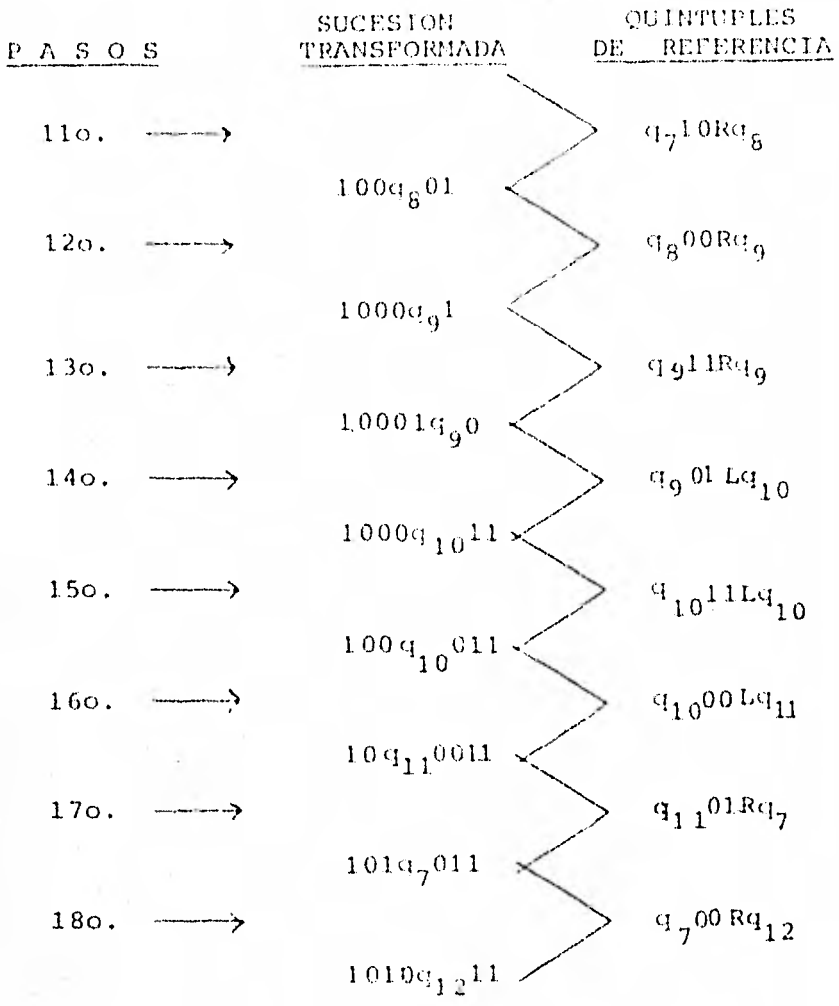
Es decir, considerando la sucesión original como una representación por marcas de un par de números enteros positivos, queremos formar la suma de estos números, a la par que conservar los números originales. En la tabla 4.3 se muestra una máquina Turing que realiza esta tarea. Suponemos que está inicialmente en el estado q_o y examina el "1" situado más a la izquierda.

	0	1
q_o	$0Rq_7$	$0Rq_1$
q_1	$0Rq_2$	$1Rq_1$
q_2	$0Rq_3$	$1Rq_2$
q_3	$1Lq_4$	$1Rq_3$
q_4	$0Lq_5$	$1Lq_4$
q_5	$0Lq_6$	$1Lq_5$
q_6	$1Rq_o$	$1Lq_6$
q_7	$0Rq_{12}$	$0Rq_8$
q_8	$0Rq_9$	$1Rq_8$
q_9	$1Lq_{10}$	$1Rq_9$
9_{10}	$0Lq_{11}$	$1Lq_{10}$
q_{11}	$1Rq_7$	$1Lq_{11}$

Tabla 4.3. Un sumador.

Para ver funcionar esta máquina, apliquémosla a la sucesión:- 101.

PASOS	SUCESION TRANSFORMADA	QUINTUPLES DE REFERENCIA
	q_0101	
1o. ⟶		q_010Rq_1
	$0q_101$	
2o. ⟶		q_100Rq_2
	$00q_21$	
3o. ⟶		q_211Rq_2
	$001q_20$	
4o. ⟶		q_200Rq_3
	$0010q_30$	
5o. ⟶		q_301Lq_4
	$001q_401$	
6o. ⟶		q_400Lq_5
	$00q_5101$	
7o. ⟶		q_511Lq_5
	$0q_50101$	
8o. ⟶		q_500Lq_6
	q_600101	
9o. ⟶		q_601Rq_o
	$1q_o0101$	
10o. ⟶		q_o00Rq_7
	$10q_7101$	



Se para la máquina.

Obsérvese que la máquina del ejemplo 4.3 realiza la tarea sin necesidad de emplear marcadores. De hecho se pueden usar, pero no es obligado; lógicamente, al introducir marcadores, la solución del problema puede hacerse usando una máquina con -- menos estados. En general, hay cierta compensación entre estados y símbolos: si no se usa un marcador para localizar un punto específico, entonces, deben usarse estados adicionales para conservar la pista de la situación de la cabeza lectora-inscriptora respecto a ese punto. Pero no podemos reducir el número de símbolos por debajo de dos.

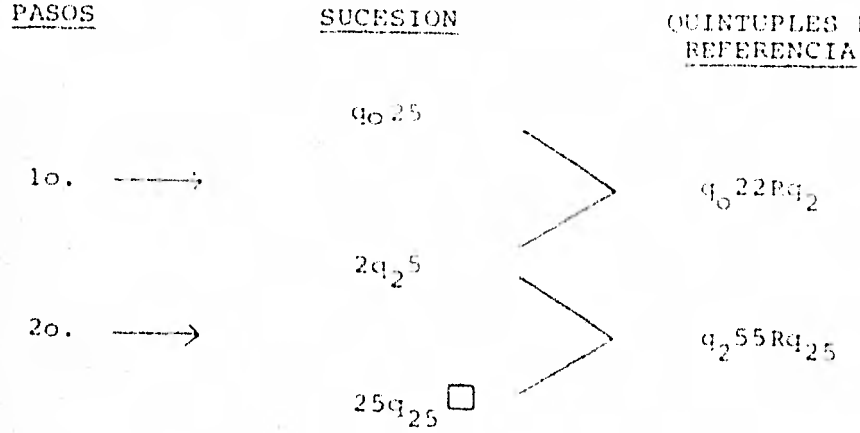
Ejemplo 4.4 UN RECONOCEDOR DE NUMEROS.

Este es un fragmento de una máquina de Turing diseñada para terminar su operación en uno de cien estados diferentes, dependientes de cuál de los números 00, 01, 02, ..., 99 está impreso sobre la cinta. La máquina se -- muestra en la Tabla 4.4. Está inicialmente en el estado q_o examinando el dígito situado más a la izquierda del par.

	0	1	2 - - - - - - 9
q_o	$0Rq_o$	$1Rq_1$	$2Rq_2$ - - - - - $9Rq_9$
q_1	$0Rq_{10}$	$1Rq_{11}$	$2Rq_{12}$ - - - - - $9Rq_{19}$
q_2	$0Rq_{20}$	$1Rq_{21}$	$2Rq_{22}$ - - - - - $9Rq_{29}$
⋮			
	- - - - - - - - - - - - - - - - - -		
q_9	$0Rq_{90}$	$1Rq_{91}$	$2Rq_{92}$ - - - - - $9Rq_{99}$

Tabla 4.4. Un reconocedor de números.

Para ver funcionar la máquina, tomemos un numeral decimal de dos dígitos, digamos el 25, y apliquémosle el mecanismo.



PARA.

Como resulta evidente por este último ejemplo, podemos describir una máquina de Turing que distinguirá entre un número finito arbitrario de numerales u otras sucesiones de símbolos, y actuará en consecuencia.

Luego, una vez que fijemos un conjunto de símbolos y un método exacto de escribir descripciones de problemas, podemos - - construir una máquina de Turing que distinguirá entre problemas y reaccionará en consecuencia, es decir, intentará resolver el problema descrito.

En particular, tenemos un método exacto para describir máquinas de Turing y los datos sobre los que operan.

Así pues, se puede construir una máquina de Turing M que aceptará como datos la descripción de una máquina de Turing arbitraria T y algunos otros datos, y efectuará sobre estos últi-

mos datos los mismos cálculos que los que habría hecho T.

Una tal máquina M se conoce como una MAQUINA UNIVERSAL DE TURING.

Lo anterior no significa que se puedan resolver todos los problemas con máquinas de Turing. Veamos porqué:

1.- Una máquina de Turing universal acepta como entrada la descripción de una máquina de Turing.

2.- Esto nos lleva a pensar en lo que sucedería si usamos la descripción de una máquina de Turing como entrada para una máquina de Turing arbitraria.

3.- En particular, si alimentamos con una descripción de una máquina T a la misma máquina T, tendremos dificultades. En realidad, se puede mostrar aún más, que ninguna máquina de Turing puede decidir si una máquina de Turing arbitraria puede desarrollar o no un cálculo cuando los datos que se le suministran son los de su propia descripción.

Un problema insoluble que podemos visualizar más fácilmente es el "problema de la parada" que discutiremos en la siguiente sección.

COMENTARIO:

Para finalizar esta sección cabe mencionar el siguiente detalle importante que nos muestra una rama de la génesis de la computación: aparte de otros modelos de máquinas de Turing, varios especialistas han propuesto y estudiado un número considerable de "máquinas" muy diferentes. Estos dispositivos se conocen, en general, con el nombre de AUTOMATAS y, muy a menudo, tienen capacidades menores que las de la máquina de

Turing. Sin embargo, el estudio de estos dispositivos proporciona una visión penetrante sobre las complejidades relativas de diferentes clases de problemas.

5.- LOS PROBLEMAS DEL CASTOR LABORIOSO Y LA PARADA.

Las máquinas de Turing están construídas para efectuar tareas específicas como, por ejemplo, la adición o la multiplicación. Parte de la construcción es la admisión tácita de un formato-estándar para la sucesión con que se le va a alimentar. Tal como se especificó en los ejemplos 4.2 y 4.3.

Pues bien, es muy natural ahora preguntarse sobre cuál sería el comportamiento de la máquina si la sucesión de entrada no fuera estándar. Es decir, ¿qué sucedería si, por ejemplo, el contador que hemos diseñado encontrase símbolos distintos de 0, 1 y a entre los signos de pesos $?.

ó, ¿qué sucedería si el signo $ de la derecha faltase?.

Esto es precisamente el "Problema de la parada" (halting problem), que podemos redactarlo así:

> "dada una máquina de Turing y una cinta arbitraria, determinar si la máquina hará eventualmente un alto o no en el uso de la cinta dada como alimentación de entrada".

Pues este y el problema con él relacionado del Castor Laborioso se ha demostrado que no se pueden resolver por ninguna máquina (o algoritmo) de Turing. Es decir, no es posible diseñar un algoritmo que resuelva este problema.

La palabra esencial aquí es "eventualmente".

Es fácil determinar si una determinada máquina se va a parar dentro de 1,479,645 o cualquier otro número determinado de pasos: simplemente, se intenta que la máquina funcione durante (1,479,645) + 1 pasos. Pero con el "eventualmente" no tenemos límite sobre el número posible de pasos que pueden ocurrir.

Vayamos con el problema relacionado del Castor Laborioso.

Como una máquina de Turing se construye partiendo sólo de conjuntos finitos (conjuntos de símbolos, estados, movimientos, quíntuples) es claro entonces que hay solamente un número finito de máquinas de Turing de un tamaño dado (es decir, de un número dado de estados y símbolos).

Por ejemplo, si tenemos:

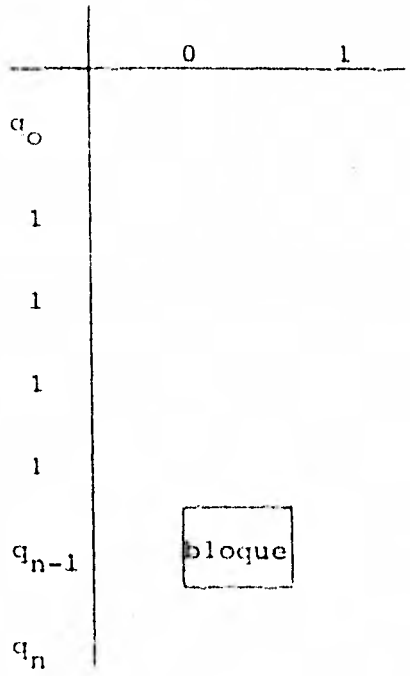
- n estados, q_0, ..., q_{n-1} (sin contar con el estado de parada q_n).
- dos movimientos, R, L

y

- dos símbolos, 0 y 1,

entonces, cada bloque en la tabla que describe una máquina puede llenarse en $2 \cdot 2 \cdot (n + 1) = 4 (n + 1)$ formas (el 1 extra es por el estado de parada q_n).

En general puede llenarse de $m \cdot 2 \cdot (n+1)$ formas, donde m son los símbolos, 2 los movimientos y n+1 los estados.



Como hay 2n bloques en la tabla, si requerimos que cada bloque se llene, hay entonces exactamente $N = [4\ (n + 1)]^{2n}$ máquinas de Turing de n-estados y 2 símbolos.

Pues bien, el "Problema del Castor Laborioso" de clase (n,2) es el de determinar cuál de estas máquinas terminará con el máximo número de unos inscritos en su cinta suponiendo que al comenzar la cinta está en blanco.

Vemos pues que éste es un problema de parada especializado y el Prof. T. Rado (E.E.U.U.) demostró en 1962 que es también insoluble. Finalizamos aquí la discusión sobre las máquinas de Turing.

Con esto, vistas las raíces del árbol "Lógico" de las computadoras (algoritmos de Markov y Máquinas de Turing), estamos preparados para dar un paso más hacia el presente "algoritmi-

co" de nuestro mundo computarizado.

6. COMPUTADORAS DIGITALES.

Tanto los algoritmos de Markov como las máquinas de Turing - son muy satisfactorios teóricamente. Se manejan en ellos - - principios fácilmente inteligibles, se puede mostrar que son- unos equivalentes a las otras y que sus relaciones con los -- sistemas de cálculo son las que se habían propuesto. Sin embargo, su propia sencillez los hace inadecuados para la mayor parte de las necesidades de cálculo, como, por ejemplo, el -- trabajo que se necesitaría para efectuar el simple producto - de dos enteros por cualquiera de estos dispositivos.

Queremos ahora examinar un artificio muy próximo a las modernas computadoras digitales y ver cómo resolveríamos los pro-- blemas con su uso.

Se hace la advertencia de que muchas consideraciones tales como las limitaciones de tamaño de los registros, problemas de- regulación de tiempo, dispositivos para la alimentación y la- emisión de resultados, etc., se omiten, aunque son vitales para la obtención de una computadora. Nuestro propósito es el- de examinar la lógica que respalda el método de solución y no el de entrar en los detalles utilizados para conseguir que -- las máquinas funcionen.

La "computadora" que discutiremos es una abstracción extremadamente simplificada de las computadoras digitales corrientes. Las computadoras reales tienen a menudo 100 códigos de operación, varios registros de índices para ayudar a la dirección, diferentes tipos de almacenamiento, y muchas complicaciones - más.

Nosotros, en cambio, supondremos una sola unidad de almacenamiento a la que nos podemos dirigir simbólicamente. Es decir, nos podemos referir a una cantidad x almacenada como "x": no necesitamos conocer su colocación exacta en el almacén.

Dos registros especiales, el acumulador y el MQ (multiplicador-cociente), se usarán para todas las operaciones. Estas no se consideran parte del almacenamiento.

Así pues, tenemos supuesto:

- una unidad de almacenamiento
- dos registros especiales
  - el acumulador
  - el MQ

También supondremos 12 instrucciones que tienen una sola dirección y caen dentro de tres categorías:

a) 4 instrucciones de transferencia

b) 4 instrucciones aritméticas

c) 4 instrucciones de control

NOTA:

Por "de una sola dirección", queremos decir que una instrucción se refiere solamente a una colocación del almacén. Digamos que es instrucción 1-aria.

Entonces, por ejemplo, en una sola instrucción no pode-- mos sumar dos números, ya que no podemos referirnos a ambos dentro de una misma instrucción.

a) Instrucciones de Transferencia.

Tratan de la transferencia de información entre el almacén y- los dos registros especiales, y se usan para mover informa- - ción; su formato es el siguiente:

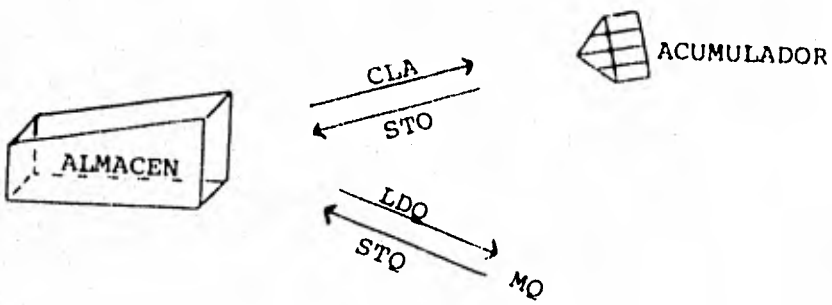
CLA: del almacén
del acumulador
("aclare y añada")

STO: del acumulador
del almacén
("guarde del acumulador"
-store the accumulator-)

LDQ: del almacén
al MQ
("cargue -Load- el MQ")

STQ: del MQ
al almacén
("guarde del MQ")

Es decir, el siguiente dibujo nos ilustra la transferencia:



Se considera que estas instrucciones son no destructivas, es-decir,

> si la cantidad X se encuentra en el almacén, entonces la ejecución de la instrucción CLAX produce una copia de X- en el acumulador, pero retiene a X en el almacén.
>
> Y si Y se encuentra en el MQ, entonces STQ Y produce una copia de Y en el almacén, aparte de la copia de Y dejada en el MQ.

Análogamente STO y LDQ.

b) Instrucciones Aritméticas.

Corresponden a la SUMA, RESTA, MULTIPLICACION y DIVISION. Su-formato respectivo es:

- ADD
- SUB
- MPY
- DIV

Como estas son instrucciones de una sola dirección y las operaciones aritméticas envuelven dos operandos, es necesario tener previamente uno de los operandos situado ya en el acumulador, o para la multiplicación en el MQ.

Esta no es una restricción considerable en la programación, - ya que, muchas veces, estas operaciones se usan sucesionalmen

te, y uno de los operandos se encuentran ya en el acumulador o el MQ por la operación anterior.

Las sucesiones básicas para producir sumas, diferencias, productos y cocientes, y colocarlos en el almacén, son las siguientes:

Y + Y = Z : CLAX, ADDY , STOZ (sólo se usa el acumulador)

Y - Y = Z : CLAX, SUBY, STOZ

X x Y = Z : LDQX, MPY Y, STQZ (ó STOZ) (El multiplicador se mantiene en el registro MQ. El producto se expande a dos veces la longitud de un registro de almacenamiento normal — por ejemplo, 76 x 24 = 1824 — y se mantiene tanto en el MQ como en el acumulador).

X ÷ Y = Z + CLAX, DIVY, STQZ, (en la división, el cociente se deja en el MQ con el residuo en el acumulador. El residuo, por tanto, también se tiene a mano, si se necesita

OBSERVACION:

En la programación para una computadora real, los procesos aritméticos se complican por el hecho de que el programador debe tener presentes las posibilidades de rebosamiento (overflow), división por cero, y muchas otras.- Sin embargo, las instrucciones básicas son las que hemos dado.

c) Instrucciones de Control

Transferencia incondicional	TRA ("Transfiérase a....")
Transferencias condicionales	TRZ ("Transfiera en cero") TRP ("Transfiera en positivo")
Instrucción de alto	HLT ("alto" — "halt")

TRAX no hace nada aritmético, pero es causa de que la instrucción siguiente se ejecute sobre la cantidad rotulada "X". La sucesión continúa luego automáticamente, desde ese -- punto, sin volver a la sucesión primitiva.

TRZ causa la Transferencia de control siempre que el contenido del acumulador es cero.

TRP causa la Transferencia de control siempre que el contenido del acumulador es positivo.

HLT hace que la computadora cese de funcionar. No requiere- ningún operando.

NOTA:

Supondremos que nos podemos dirigir al acumulador y al - MQ por los nombres "ACC" y "MQ", respectivamente.

Por ejemplo:

LDQ ACC causaría que el contenido del acumulador pasase al MQ.

Pero no se permite transferir control ni al acumulador ni al-MQ.

Aunque una computadora real tiene un repertorio mucho más - - grande de instrucciones y es por ello capaz de efectuar los - cálculos de un modo comparativamente más sofisticado, veremos unos ejemplos sobre nuestra elemental "computadora" que ilustran el modo en que uno realizaría algoritmos sobre una computadora digital.

Ejemplo 6.1

El problema es el de resolver el sistema de dos ecuaciones lineales en x y y:

$$\begin{cases} a\,x + b\,y = c \\ d\,x + e\,y = f \end{cases} \quad ; \; ae - bd \neq 0$$

Suponemos que los coeficientes están almacenados como A, B, - C, D, E y F, y que los resultados se llamarán X y Y en la computadora. En la práctica real, necesitaríamos leer los coeficientes en el almacén e imprimir los resultados; pero aquí ignoramos la necesidad de estos pasos.

La solución de estas ecuaciones está dada por las ecuaciones:

$$\begin{cases} x = \dfrac{ce - bf}{ae - bd} \\ y = \dfrac{af - cd}{ae - bd} \end{cases}$$

En el programa abajo dado hemos supuesto que los coeficientes son tales que las partes significativas de los productos aparecen en MQ.

Los comentarios que aparecen a la derecha de algunas líneas del programa no forman parte del mismo, son sólo explicaciones al procedimiento.

```
LDQ   A
MPY   D
STQ   TEMP    almacénese el producto ae y llámesele TEMP.
LDQ   B
MPY   D
CLA TEMP      En este momento bd está en el MQ y ae en el-
              acumulador.
SUB   MQ
STO   DENOM   El denominador ae-bd ha sido calculado.
```

```
LDQ   C       con esto comienza el cálculo para x.
MPY   E
STQ TEMP      TEMP contiene ahora el producto ce.
LDQ   B
MPY   F
CLA TEMP
SUB   MQ      esto calcula el numerador para x.
DIV DENOM
STQ   X
```

```
LDQ   A        con esto comienza el cálculo de y.
MPY   F
STQ   TEMP
LDQ   C
MPY   D
CLA TEMP
SUB   MQ       el numerador para y.
DIV   DENOM
STQ y
```

```
HLT            se para el proceso.
```

Como puede observarse, el programa consta de tres partes, - que son casi iguales, excepto por los datos usados. Siempre- que el cálculo en varias partes del programa es el mismo, pero aplicado a datos diferentes, es conveniente hacer que la - computadora repita un ciclo y no tener que escribirlo de nuevo gastando espacio de almacenaje para varias copias del mismo.

En el ejemplo siguiente programamos el algoritmo para determinar si una fórmula polaca está o no bien formada, usando la - capacidad del computador para repetir un ciclo varias veces.- La carta de flujo para este algoritmo se dio en la figura - - 2.3

Ejemplo 6.2

Suponemos que tenemos en el almacén los datos: NUM, S(1), S(2),..., S(NUM), y las constantes: C, A, K, E y N.

Nótese que los datos, en este caso, son alfabéticos. Sin embargo, en la computadora están representados en la misma forma que si fueran datos numéricos (e instrucciones), y de aquí que podemos efectuar operaciones numéricas sobre ellos.

Necesitamos también en el almacén la constante numérica 1.

Las letras que aparecen a la izquierda de algunas líneas, son etiquetas de identificación a dichas líneas, y sí forman parte del programa.

```
        CLA   NUM
        STO   1
AA      CLA   S(I)
        SUB   N
        TRZ   AB     Transfiérase a instrucción AB, si S(I) = N.
        ADD   N
        SUB   C
        TRZ   AC     Transfiérase a la instrucción AC, si S(I)=
                     C.
        ADD   C
        SUB   A
        TRZ   AC     Transfiérase, si S(I) = A.
        ADD   A
        SUB   K
        TRZ   AC     Transfiérase, si S(I) = K.
        ADD   K
```

```
        SUB   E
        TRZ   AC      Ninguna Transferencia, si S(I) es una va--
                      riable proposicional.
        CLA   I
        SUB   NUM
        TRZ   AD      Transfiérase a AD, si I = NUM (condición -
                      inicial).
        CLA   I
        ADD   ONE     ONE es la locación de la constante "1".
        STO   J
        CLA   SUM (J)
        ADD   ONE
AE      STO   SUM (I)
        CLA   ONE
        SUB   I
        TRZ   AF      Transfiérase al final del proceso.
        CLA   SUM(I)
        TRP   AG      Transfiérase, si la suma parcial es acepta
                      ble.
AH      HLT           este es el final si la fórmula no está -
                      bien formada.
AB      CLA   I       meta esta instrucción, si (S(I) = N.
        SUB   NUM
        TRZ   AH      Transfiérase, si el último símbolo es N.
        CLA   I
        ADD   ONE
```

```
        STO  J
        CLA  SUM(J)
        TRA  AE      Váyase a AE
AC      CLA  I       meta esta instrucción, si S(I) es C,A,K ó-
                     E.
        SUB  NUM
        TRZ  AH      Transfiérase, si el último símbolo es C,A,
                     K ó E.
        CLA  I
        ADD  ONE
        STO  J
        CLA  SUM(J)
        SUB  ONE
        TRA  AE      Transfiérase a AE.
AD      CLA  ONE
        TRA  AE      Transfiérase a AE.
AF      CLA  SUM(ONE) se termina el proceso
        SUB  ONE
        TRZ  AI
        TRA  AH
AG      CLA  I
        SUB  ONE
        STO  I
        TRA  AA
AI      HLT  esto es el fin, si la fórmula está bien formada.
```

En este programa,por el uso de subíndices como en SUM (I) - (Σi en la carta de flujo) hemos supuesto que la computadora- tenía la capacidad de localizar información dándole ese tipo- de dirección. Podría haberse mejorado el programa, si hubié- semos incluido algunas otras instrucciones, tales como una -- transferencia sobre negativos y una transferencia sobre no ce ro. También se necesitaba haber puesto instrucciones para ob tener los mensajes que aparecen en la carta de flujo.

Ejemplo 6.3

El problema consiste en arreglar en orden alfabético una serie de N palabras.

El convenio es que la palabra X precede a la palabra Y - si y sólo si como números en la computadora X es más pe- queño que Y.

El proceso de ordenación usado es un intercambio. La - primera palabra se compara con todas las otras. Siempre que se encuentra una palabra que la precede se hace un - intercambio, y la investigación continúa con la nueva -- primera palabra. De esta forma la primera palabra, en - el orden alfabético, se convierte en la primera palabra- de la serie. El procedimiento se sigue aplicando luego- a la segunda palabra, a la tercera, etc....

Las palabras se encuentran en los lugares: W(1) a W(N),- y N está almacenado, como también lo está la constante - ONE.

```
    CLA  ONE
    STO  I
AF  ADD  ONE
    STO  J
```

```
AA  CLA  W(I)
    SUB  W(J)
    TRP  AB      Transfiérase para intercambiar las palabras
                 W(I) y W(J)
AD  CLA  J
    ADD  ONE
    STO  J
    SUB  N
    TRP  AC      Transfiérase, si W(I) ha sido comparada con
                 todas las otras palabras.
    TRA  AA      Váyase a AA.
AB  CLA  W(J)    intercámbiense W(I) y W(J)
    LDQ  W(I)
    STO  W(I)
    STQ  W(J)
    TRA  AD      Transfiérase a AD.
AC  CLA  I
    ADD  ONE
    STO  I
    SUB  N
    TRP  AE      Transfiérase al final.
    CLA  I
    TRA  AF      Transfiérase a AF.
AE  HLT          Termina el proceso.
```

7. LENGUAJES DE PROGRAMACION

Esta sección complementa lo visto en las precedentes con un corto comentario del PORQUE existen lenguajes de programación, poniendo de manifiesto su ventaja y utilidad en contraste con el aparente "aumento de trabajo" al añadir con estos un prerequisito o traba más para el trato con la computadora. Aludiendo a los ejercicios últimos, veremos que de hecho tales prerequisitos, lejos de ser una traba, son un puente hacia la computadora que nos libra de un río de confusión y errores.

En la sección anterior dimos tres ejemplos de algoritmos o programas escritos para una computadora digital muy primitiva. Pueden escribirse máquinas de Turing o algoritmos de Markov que realicen las mismas tareas, pero se llevarían muchas páginas y lo más seguro es que la persona que los escribiese cometería muchos errores antes de que hiciera la versión correcta. No hay nada más fácil en la Teoría de algoritmos que equivocarse. Incluso el lenguaje de la computadora que usamos en la última sección es lo suficientemente primitivo para que sea muy fácil cometer errores al escribir un programa largo.

En un esfuerzo para disminuir el número de errores y poder hacer programas que se puedan entender más rápidamente por quien los examine, han surgido un cierto número de lenguajes de programación, y con ellos traductores o algoritmos (llamados compiladores), que hacen que el computador pueda "leer" los lenguajes.

Cada uno de estos tiene sus propias reglas gramaticales de construcción y están diseñados para desarrollar diferentes tipos de labores. Nuestro propósito no es ahorita tomar un determinado lenguaje y emplear otro capítulo en su estudio, sino poner de relieve lo lógico y necesario de su aparición en la evolución de las máquinas digitales.

Entre los lenguajes más conocidos están:

	FORTRAN ALGOL BASIC MAD IOVIAL PASCAL	destinados primariamente para cálculos científicos,
y también	COBOL BASIC EXT.	para cálculos en los negocios,
	COMIT SNOBOL RPG LISP IPL-V	para el procesamiento de listas de información;

y más recientemente, el PL/I, que tiene algunas de las características de cada uno de los anteriores.

Aunque no es nuestro propósito aquí la programación, es útil con fines de comparación, ver ejemplos de algoritmos escritos en un lenguaje de programación.

Para estos ejemplos se ha escogido como lenguaje el BASIC. Los ejemplos son los mismos que los usados en la sección previa, y los programas son fácilmente legibles, pues se usa un BASIC muy simplificado. (Hay otras versiones de BASIC ya muy sofisticadas, desarrolladas sobre todo por la Cromemco Inc.,-

Mountain View, California).

Ejemplo 7.1

La solución de dos ecuaciones lineales (véase el Ejemplo 6.1).

Ejemplo 7.2

Fórmulas Bien Formadas. (Véase el ejemplo 6.2)

Observación al ejemplo 7.2:

En las líneas de comentarios REM, las etiquetas AA, AB - AC, AD, AE y AH son las que están como referencias en el ejemplo 6.2 y en la Carta de Flujo 2.3.

Ejemplo 7.3

Una sencilla ordenación. (Véase el ejemplo 6.3)

En las páginas siguientes se dan los listados y "corridas" -- respectivas de los programas. En los ejemplos 7.1 y 7.3 sólo se da una corrida por cada ejemplo; y en el 7.2 se dan varias que ilustran la alternativa de ser o no ser f.b.f.

```
100 REM PRO.EJ. EJEMPLO 4.1
200 PRINT"SOLUCION DE DOS ECUACIONES LINEALES"
300 PRINT"          CON DET. NO NULO"
400 READ A,B,C,D,E,F
500 LET W=(A*D-B*C)
600 LET X=(C*E-B*F)/W
700 LET Y=(A*F-C*D)/W
800 PRINT
900 PRINT"X=";X
1000 PRINT"Y=";Y
1100 DATA 1.2,2.3,3.4,4.5,5.6,6.1
1200 PRINT
1300 PRINT"PARA OTRO PAR DE ECS. VAYA LINEA 1100"
1400 END
#
```

```
#
#
#
#
#
#
R;FILE FILE1(PRINTER)
#RUNNING 6460
SOLUCION DE DOS ECUACIONES LINEALES
        CON DET. NO NULO

X= 1.2801652893
Y= 2.1983471074

PARA OTRO PAR DE ECS. VAYA LINEA 1100
#ET=13.2 PT=0.2 IO=0.5
```

```
170   IF
180   NEXT J
190   IF
200
210   REM
220   IF
230
240
250   GOTO
260   IF
270
280   IF
290   GOTO 240
300   REM
310
320
330   DATA
340   DATA
350
```

```
4210 GOTO 4220
```

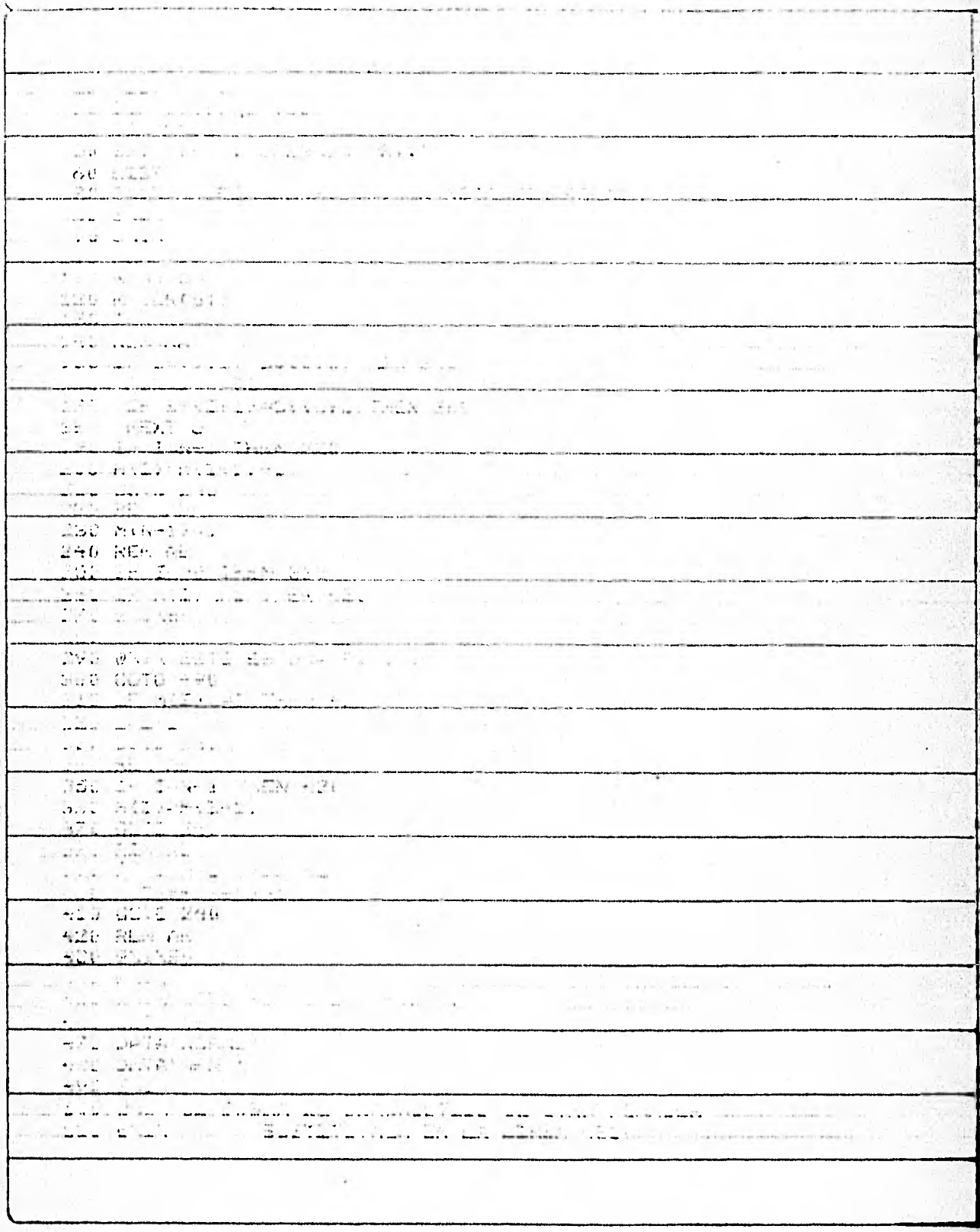
```
160   NEXT J
240   REM RE
300 GOTO 
420 GOTO 240
420   REM 
```

[illegible] F.B.F. [illegible]

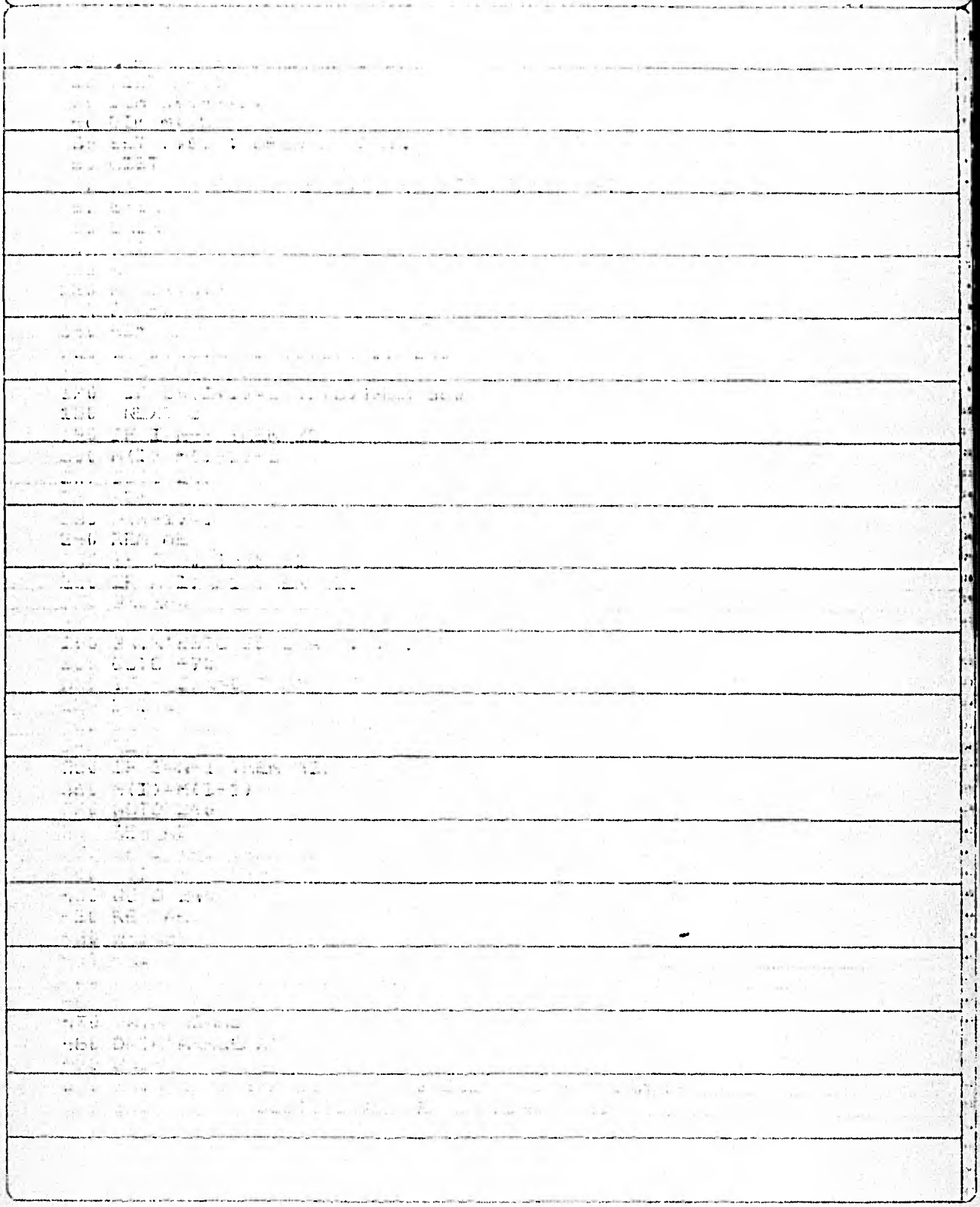
NOMBRE

ESTO NO ES UNA F.B.F.

[illegible]



ESTO ES UNA F.B.F.

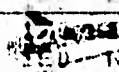


```
50 SET [illegible]
60 LIST
[illegible]
110 READ [illegible]
120 N=LEN(A$)
130 T=N-1
[illegible]
170  IF S$(I,I)=C$(J,J) THEN 380
180  NEXT J
190 IF I=N-1 THEN 220
[illegible]
230 M(N-1)=1
240 REM AE
[illegible]
290 @\1\ ESTO ES UNA [illegible]
300 GOTO 490
310 IF M(I)<=0 THEN 420
[illegible]
350 IF I=N-1 THEN 420
360 M(I)=M(I+1)
370 GOTO 340
[illegible]
410 GOTO 240
420 REM AH
430 @[illegible]
[illegible]
470 DATA"INCAN[illegible]
480 DATA"AKKADIAN[illegible]
490 @\[illegible]
500 [illegible]
```

[illegible] DE F.I.F. EN NOTACION POLACA

[illegible]

ESTO ES UNA F.B.F.



260

```
170  IF S$(I,J)=S$(I,0)THEN 200
180  NEXT J
190 IF I=N-1 THEN 230
200 REM
210 REM
220 REM
230 M(N-1)=1
240 REM AE
250 IF I=0 THEN 300
260 IF M(I)=1 THEN 290
270 REM
280 REM
290 PRINT"ESTO ES UNA PALABRA"
300 GOTO 490
310 IF I=0 THEN 340
320 REM
330 REM
340 IF I=N-1 THEN 420
360 M(I)=M(I+1)
370 GOTO 240
380 REM AE
390 REM
400 M(I)=M(I+1)+1
410 GOTO 240
420 REM AE
430 PRINT
440 REM
450 REM
460 DATA PLAZA
470 DATA CLINICA
480 REM
490 REM
500 REM                    PALABRA EN LA LINEA 460
```

```
10 REM AB
20 LIST"PRN:"
30 DIM C$(9),S$(20)
40 DIM M(19)
50 SET 0,80 : OPEN\1,"PRT:"
60 LIST
70 @\1\" PRUEBA DE F.B.F. EN NOTACION POLACA"
80 @\1\
90 @\1\
100 REM C9
110 READ S$
120 N=LEN(S$)
130 I=N-1
140 REM AB
150 IF S$(I,I)=C$(J,J)THEN 360
160 FOR J=1 TO 4
170 IF S$(I,I)=C$(J,J)THEN 360
180 NEXT J
190 IF I=N-1 THEN 220
200 M(I)=M(I+1)+1
210 GOTO 230
220 REM AD
230 M(N-1)=1
240 REM AE
250 IF I<>0 THEN 310
260 IF M(I)<>1 THEN 420
270 @\1\S$
280 @\1\
290 @\1\"ESTO ES UNA F.B.F."
300 GOTO 490
310 IF M(I)<=0 THEN 420
320 I=I-1
330 GOTO 140
340 REM AE
350 IF I=N-1 THEN 420
360 M(I)=M(I+1)
370 GOTO 240
380 REM AC
390 IF I=N-1 THEN 420
400 M(I)=M(I+1)-1
410 GOTO 240
420 REM AH
430 @\1\S$
440 @\1\
450 @\1\"ESTO NO ES UNA F.B.F."
460 GOTO 490
470 DATA"NCARE"
480 DATA"KPNCAPO"
490 @\1\
500 @\1\"SI DESEA UNA VERIFICACION DE OTRA FORMULA
510 @\1\"        SCRIBA 'RUN' EN LA LINEA 10"
```

PRUEBA DE F.B.F. EN NOTACION POLACA

ESPAÑOL

ESTO NO ES UNA F.B.F.

~~SI DESEA EL DIAGNOSTICO DE OTRA FORMULA~~
SUSTITUYALA EN LA LINEA 4120

```
#
L
100   REM VER EL EJEMPLO 4.3
200   PRINT "SORT"
300   DIM W(15)
400   READ N
500   REM N>1,ENTERO.
600   FOR I=1 TO N
700   READ W(I)
800   NEXT I
900   LET I=1
1000   LET J=I+1
1100   IF W(I)<=W(J) THEN 1500
1200   LET A=W(I)
1300   LET W(I)=W(J)
1400   LET W(J)=A
1500   IF J=N THEN 1800
1600   LET J=J+1
1700   GO TO 1100
1800   IF I=N-1 THEN 2200
1900   LET I=I+1
2000   GO TO 1000
2100   DATA 7,14,18,11,2,7,9,23
2200   FOR I=1 TO N
2300   PRINT W(I)
2400   NEXT I
2500   PRINT "SI DESEA ORDENAR OTRA  SECUENCIA"
2600    PRINT "MODIFIQUE LA LINEA 2100"
2700    END
#
```

```
#
#
#
#
R:FILE FILE1(PRINTER)
#RUNNING 6509
SORT
 2
 7
 9
 11
 14
 18
 23
SI DESEA ORDENAR OTRA  SECUENCIA
MODIFIQUE LA LINEA 2100
#ET=15.8 PT=0.2 IO=0.9
```

C) LENGUAJES FORMALES.

1. Lenguajes de Post.

2. Avances recientes en los lenguajes formales.

1.- LENGUAJES DE POST

Virtualmente, toda la actividad asociada con los conceptos discutidos en el capítulo, es de un carácter general:

> la actividad del cambio formal de sucesiones de-símbolos por otras sucesiones de símbolos mediante la aplicación de ciertas reglas prescritas.

Una de las primeras personas en reconocer esto fue el lógico-americano Emil Post.

En esta sección examinaremos la teoría que él desarrolló y dió a conocer en 1943 y que se conoce actualmente como la "Teoría de los Lenguajes de Post".

- Nos vamos a ocupar de operaciones sobre "un conjunto finito no vacío de símbolos", llamado "el alfabeto". Estos símbolos se corresponden con el conjunto de variables y conectivos en el cálculo proposicional (junto con los paréntesis) o con el alfabeto del español habitual.

- Hay "un conjunto finito de reglas" que se aplican a estos símbolos y que especifican cómo combinar sucesiones de símbolos en palabras o sentencias (las f.b.f. de la lógica o las palabras del castellano). A estas reglas se les llama "producciones".

- Generalmente tenemos también un conjunto de sucesiones, que llamamos "AXIOMAS" a las que podemos aplicar inicialmente las producciones.

Pasamos ahora a definir formalmente todos estos términos.

Definición 1.1

- Sea $\mathcal{A}$ un conjunto finito no vacío llamado el ALFABETO.
- Sea $\mathcal{V}$ el conjunto de todas las sucesiones finitas de símbolos de $\mathcal{A}$, conjunto al que llamaremos el VOCABULARIO.
- Supongamos que los símbolos $\alpha_1, \alpha_2, \alpha_3, \ldots\ldots$, no están en $\mathcal{V}$. (informalmente, usaremos: $\alpha, \beta, \gamma, \ldots$)

• Un ESQUEMA SUCESIONAL es una sucesión finita:

$\alpha_{i_1} s_1 \alpha_{i_2} s_2 \ldots\ldots \alpha_{i_{n-1}} s_{n-1} \alpha_{i_n}$

- cada s_i es un elemento de $\mathcal{V}$

y

- al menos una de las α aparece realmente (una o más pueden ser nulas o no presentarse).

• Una PRODUCCION es una sucesión:

$s_1, s_2, \ldots\ldots, s_m \longrightarrow T$, donde

- m es un entero positivo.
- $s_1, s_2, \ldots\ldots, s_m$ y T son esquemas sucesionales.

y

- si $T = \alpha_1 T_1 \alpha_2 T_2 \ldots\ldots \alpha_{n-1} t_{n-1} \alpha_n$, entonces al menos una de las T_i no es la sucesión nula y cada una de las α_i que ocurre realmente en T.- también ocurre al menos en una de las s_j, $j = 1, \ldots, m$.

El uso de las producciones es análogo al de las producciones de Markov, pero sin la orientación de izquierda a derecha. Un esquema sucesional se piensa como representando el conjunto de todas las sucesiones que pueden obtenerse mediante la substitución de los α de $\mathcal{V}$ por sucesiones arbitrarias. Tales sucesiones puede decirse que satisfacen el esquema.

> Si entonces tenemos en un conjunto dado de sucesiones algunas sucesiones que satisfacen $s_1, \ldots, s_m$, donde tenemos una producción $s_1, \ldots, s_m \longrightarrow T$, podemos añadir al conjunto la sucesión que satisface a T, que se obtiene substituyendo a las α de T por las sucesiones que corresponden a estas mismas α en las $s_i, \ldots, s_m$. (Suponemos aquí que si una de las α aparece en dos o más de las s_j se reemplazará en todas sus ocurrencias por la misma sucesión).

Ejemplo 1.1

Supongamos que tenemos dos producciones:

$$\alpha\, a \longrightarrow b\, a\, \alpha$$

$$\beta\, a\, \delta\, b,\ b\, \alpha\, a\, \beta \longrightarrow \alpha\, \beta\, bb\, \delta$$

y que el conjunto dado de sucesiones se compone de la sola sucesión a. Es decir $\mathcal{V} = \{ a \}$

- Como ésta es una sucesión que satisface α a (con α reemplazada por la sucesión nula) podemos usar la primera producción para añadir al conjunto la sucesión ba.

- Esta nueva sucesión satisface b α a β , pero no tene-

mos sucesión alguna que satisfaga β_1 a δ b, de modo que aún no podemos usar la segunda producción.

- Pero ba satisface α a (con b como α), de forma que podemos usar la primera producción para añadir al conjunto otra sucesión, la bab.

- Esta sucesión ya no satisface a αa, pero satisface tanto a β_1 a δ b (con b como β_1, y δ nula) como a b α a β (con α nula y b como β). Luego podemos producir la sucesión bbb.

- Podemos también usar bab para satisfacer β_1 a δ b, y ba para satisfacer b α a β, produciendo la sucesión bb.

- Como no hay más formas de satisfacer los esquemas de sucesiones, el conjunto final de palabras es $\{a, ba, bb, bab, bbb\}$.

NOTA:

Aunque el conjunto generado en el ejemplo es finito, no es éste siempre el caso; en realidad, en las situaciones más interesantes, el conjunto generado tiene infinitos miembros.

Definición 1.2

Un LENGUAJE CANONICO DE POST, consiste en:

- un alfabeto finito
- un conjunto finito de producciones, y
- un subconjunto finito del vocabulario, llamado el conjunto de AXIOMAS.

NOTA CURIOSA:

Como el alfabeto mismo es un subconjunto finito del vocabulario, puede entonces usarse como conjunto de axiomas.

El ejemplo que sigue nos proporciona un ejemplo bonito e importante de lo que es un Lenguaje Canónico de Post, a saber: El cálculo proposicional, usando la notación polaca.

Ejemplo 1.2

El cálculo proposicional (en notación polaca) visto como un lenguaje canónico de Post.

- Debemos tener símbolos para las variables proposicionales y los conectivos, y varios símbolos de clase para distinguir las variables proposicionales, las f.b.f. y los teoremas.

- Como deseamos tener un número ilimitado de variables proposicionales, mientras que el alfabeto ha de ser forzosamente finito, debemos generar las variables proposicionales.

• Tomamos como alfabeto $\{ N, C, A, k, E, p, 1, \vdash, \mathbb{W}, \mathbb{L} \}$ los últimos tres símbolos de la lista designarán:

$\vdash$ la clase de los teoremas,

$\mathbb{W}$ la clase de las fórmulas bien formadas (f,b,f,)

$\mathbb{L}$ la clase de las variables proposicionales (letras).

• Vamos a tener solamente un axioma: $\mathbb{L}\ p$

y

- Catorce producciones:

1. $\mathbb{L}\ \alpha \longrightarrow \mathbb{L}\ \alpha\ 1$
2. $\mathbb{L}\ \alpha \longrightarrow \mathbb{W}\ \alpha$
3. $\mathbb{W}\ a \longrightarrow \mathbb{W}\ N\ \alpha$
4. $\mathbb{W}\ \alpha\ ,\ \mathbb{W}\ \beta \longrightarrow \mathbb{W}\ C\alpha\beta$
5. $\mathbb{W}\ CN\alpha\beta \longrightarrow \mathbb{W}\ A\alpha\beta$
6. $\mathbb{W}\ A\alpha\beta \longrightarrow \mathbb{W}\ C\ N\ a\ \beta$
7. $\mathbb{W}\ NC\alpha N\beta \longrightarrow \mathbb{W}\ K\ a\ \beta$
8. $\mathbb{W}\ Ka\beta \longrightarrow \mathbb{W}\ N\ C\alpha N\beta$
9. $\mathbb{W}\ N\ C\ C\ a\ \beta\ N\ C\ \beta\ a \longrightarrow \mathbb{W}\ E\alpha\beta$
10. $\mathbb{W}\ E\alpha\beta \longrightarrow \mathbb{W}\ N\ C\ C\alpha\beta\ N\ C\ \beta\ a$
11. $\mathbb{W}\alpha\ ,\ \mathbb{W}\beta \longrightarrow \vdash C\alpha C\beta\alpha$
12. $\mathbb{W}\ \alpha\ ,\ \mathbb{W}\ \beta\ ,\ \mathbb{W}\ \mu \longrightarrow \vdash CC\alpha C\beta\mu CC\alpha\beta C\alpha\mu$
13. $\mathbb{W}\alpha\ ,\ \mathbb{W}\beta \longrightarrow \vdash CC\ N\alpha\ N\beta\ CC\ N\alpha\beta\alpha$
14. $\vdash \alpha\ ,\ \vdash C\alpha\beta \longrightarrow \vdash \beta$

Con el axioma dado, las producciones especifican:

1. - La 1., que las variables proposicionales son todas sucesiones de la forma: p, p1, p11, p111,.....

2., 3., 4. - Las 2., 3., y 4., son las reglas para la construcción de fórmulas bien formadas.

5.
6.
7. - de la 5., a la 10., definen A, K y E en términos de-
8. C y N.
9.
10.

11.
12. - las 11., 12. y 13., son los "axiomas" del cálculo -
13. proposicional en el sentido del capítulo III, a).

14. - la 14. es la que nos proporciona el Modus Ponens.

Definición 1.3

El conjunto de TEOREMAS de un lenguaje canónico de Post-consiste en los axiomas del lenguaje junto con todas las sucesiones generadas, partiendo de los axiomas por las - producciones del lenguaje.

Definición 1.4

Dado un lenguaje (canónico de Post) $\mathcal{L}$, se dice que un - lenguaje $\mathcal{L}'$ es una EXTENSION del $\mathcal{L}$ si:

i) el alfabeto de $\mathcal{L}$ está contenido en el alfabeto - de $\mathcal{L}'$

y

ii) cada teorema de $\mathcal{L}$ es un teorema de $\mathcal{L}'$.

COMENTARIO:

Podemos extender un lenguaje añadiéndole más símbolos, - producciones y axiomas, pero no es necesario que la ex-- tensión se haga de esta manera. De hecho, una extensión

de un lenguaje dado puede derivarse de un modo muy distinto, incluso hasta el punto de que sea difícil reconocer el hecho de que es una extensión del lenguaje.

Definición 1.5

Una extensión $\mathcal{L}'$ de un lenguaje $\mathcal{L}$ se dice que es CONSERVATIVA, si todo teorema de $\mathcal{L}'$ que es una sucesión en el alfabeto de $\mathcal{L}$, es también un teorema de $\mathcal{L}$.

Es decir, una extensión conservativa de un lenguaje es una en que no se aumenta el número de teoremas que pueden enunciarse en el alfabeto del lenguaje original; cualesquiera nuevos teoremas contienen símbolos que no están en el alfabeto original.

Definición 1.6

Un lenguaje canónico de Post se llama NORMAL si:

i) Tiene solamente un axioma

y

ii) Todas las producciones son de la forma $s_1\alpha \longrightarrow \alpha s_2$

donde s_1 y s_2 son sucesiones especificadas.

Post mismo estableció el importante resultado de que:

"Todo lenguaje canónico tiene una extensión normal conservativa".

La prueba, que es muy larga, envuelve la construcción de una serie de lenguajes, cada uno de los cuales se muestra ser una extensión conservativa del anterior, y el último de los cuales es normal.

Este resultado proporciona entonces una forma estándar o normal para el estudio de los lenguajes de Post. Se hace la - - aclaración que, como es el caso con los sistemas lógicos, no- es ésta la única forma estándar posible.

Vemos un ejemplo final.

Ejemplo 1.3

Consideremos el siguiente lenguaje:

Alfabeto: a, b, c, d

Axiomas: I.- a
II.- bc

Producciones: 1.- α a $\longrightarrow$ ba α

2.- β a δ b, b α a β $\longrightarrow \alpha\beta$ bb δ

3.- α a β b $\longrightarrow \alpha$ cb β

4.- α c $\longrightarrow$ da α b

5.- α b $\longrightarrow$ dc α a

Pruébese que las siguientes sucesiones son teoremas de este - lenguaje:

i) dcbba

ii) badcba

i) Prueba:

1. a - - - Ax, I

2. ba ---- 1 con Prod. 1., α vacío.

3. bab --- 2 con Prod. 1, α = b

4. bbb --- 3 con Prod. 2, α y λ vacíos, $\beta = \beta_1 = b$

5. dcbba --- 4 con Prod. 5, $\alpha = bb$.

Q.E.D.

Nota:

Las abreviaturas Ax. y Prod. significan: Axioma y Producción, respectivamente.

ii) Prueba:

1. a ------ Ax. I

2. ba ----- 1 con Prod. 1, α vacío.

3. bab ---- 2 con Prod. 1, $\alpha = b$

4. dcbaa --- 3 con Prod. 5, $\alpha = ba$

5. badcba -- 4 con Prod. 1, $\alpha = dcba$.

Q.E.D.

Como se sugirió al principio del Capítulo, los varios tópicos que hemos discutido son expresables en lenguajes de Post. Por ello, sería lógico esperar encontrarse con problemas insolubles en los lenguajes de Post, exactamente como el problema del alto en las máquinas de Tuning y otros problemas que aparecieron en nuestras discusiones. Estos toman la forma de problemas de decisión sobre teoremas: decidir si una expresión dada es o no un teorema en un lenguaje dado.

2.- AVANCES RECIENTES EN LOS LENGUAJES FORMALES.

Con la llegada de las grandes computadoras digitales, se aceleraron enormemente los estudios de lenguajes formales como -

efecto de los desarrollos en dos distintas áreas:

- Por un lado, los intentos de producir algoritmos de traducción mecánica eficientes mostraron que los estudios lingüísticos tradicionales de las lenguas naturales describían estas lenguas de un modo inadecuado para estos propósitos. Hubo pues, necesidad de formalizar las propiedades de varias lenguas naturales que fueran adecuadas para la construcción de tales algoritmos.

- Por otra parte, los sofisticados lenguajes de programación que se han ido desarrollando al mismo tiempo que las computadoras, son, en sí mismos, formales en naturaleza. A medida que estos lenguajes se han hecho más flexibles, los traductores necesarios para producir programas para la máquina en estos lenguajes se han hecho más complejos. Para describir estos nuevos lenguajes adecuadamente y para construir traductores que produzcan rápidamente programas eficientes, se ha hecho necesario estudiar las características de los lenguajes formales.

En la descripción de lenguajes formales el más importante desarrollo actual ha sido la Forma Normal Backus. Consiste ésta en una formalización de una porción de metalenguajes que permite una descripción concisa y precisa del lenguaje formal.

La simbología de la Forma Normal Backus es la siguiente:

- las clases de conjuntos de sucesiones de símbolos se representan mediante el uso de paréntesis angulares, Todo lo que aparezca en el interior de tales paréntesises el nombre para una clase de sucesiones de símbolos.

- el símbolo ::= se usa como el símbolo definitorio: laclase cuyo nombre aparece a la izquierda de este símbolo

consiste en aquellas sucesiones de símbolos que aparecen enumerados a la derecha.

- Un pequeño segmento vertical, | , se usa como separador en las definiciones, y la concatenación en una proposición definicional denota concatenación dentro del lenguaje formal.

Ejemplo 2.1

Las f.b.f. del cálculo proposicional pueden describirse con las definiciones siguientes de la Forma Normal Backus, suponiendo solamente cuatro variables proposicionales.

⟨ símbolo ⟩ ::= p | q | r | s | ⊃ | ~ | (|)

⟨ variable ⟩ ::= p | q | r | s

⟨ conectivo ⟩ ::= ⊃ | ~

⟨ f.b.f. ⟩ ::= ⟨ variable ⟩ | ~ (⟨ f.b.f. ⟩) | (⟨ f.b.f. ⟩) ⊃ (⟨ f.b.f. ⟩)

- la primera de estas expresiones afirma que hay ocho símbolos que incluyen los paréntesis (,).

- la segunda define la clase de las variables.

- la tercera define la clase de los conectivos.

- la proposición final afirma que una f.b.f. es: o cualquier variable; o los símbolos ~ (seguidos, en ese orden, por cualquier f.b.f., seguida del paréntesis); o el símbolo (, seguido de cualquier f.b.f., seguida por la secuencia de símbolos) ⊃ (, seguida por cualquier f.b.f., seguida por).

Ejemplo 2.2

Estas expresiones definen la construcción de los números decimales.

⟨símbolo⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | .

⟨dígitos⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

⟨entero⟩ ::= ⟨dígito⟩ | ⟨dígito⟩ ⟨entero⟩

⟨número decimal⟩ ::= ⟨entero⟩ | ⟨entero⟩ . | ⟨entero⟩ . ⟨entero⟩

- la primera y segunda expresiones no requieren explicación.

- la tercera define un entero como un dígito, o un dígito seguido por un entero previamente definido. Es decir, un entero se define como una sucesión finita de dígitos.

- la cuarta proposición introduce el punto decimal. Nótese que sucesiones tales como 0 y 000 297 son enteros completamente legítimos, de modo que 0.000297 es un número decimal. Sin embargo, .000297 no es un número decimal deacuerdo con estas definiciones.

La dirección en que han progresado los estudios de los lenguajes formales ha sido determinada en gran parte por el trabajo de Noam Chomsky y otros, que comenzó alrededor del año 1955 o 1957.

De importancia clave en estos estudios ha sido el reconocimiento de la importancia de la estructura de la frase y el contexto en la comprensión del significado.

Definición 2.1

- Una GRAMATICA DE LA ESTRUCTURA DE FRASES es un conjunto finito de producciones de Post que contienen exactamente un símbolo S que aparece solamente a la izquierda de " ⟶ ", y un conjunto no vacío de símbolos que aparecen solamente a la derecha de "⟶ ".

- Un LENGUAJE DE ESTRUCTURA DE FRASE es un lenguaje canónico de Post cuyas producciones constituyen una gramática de la estructura de frases, y cuyo solo axioma es el símbolo S, que aparece solamente al lado izquierdo de las producciones.

Esta definición está motivada por la descripción de las formas en que se puede construir una frase.

Podemos pensar que el símbolo S significa "sentencia", y que las producciones describen las formas en que podemos construir las sentencias. Los símbolos que aparecen solamente al lado derecho de las producciones, conocidos como SIMBOLOS TERMINALES, desempeñan entonces el papel de las palabras de que se compone una sentencia.

Ejemplo 2.3

Alfabeto: S, NP, VP, AP, N, V, Adj, Adv, Art. Pred, un, una, pelota, incoloro, incolora, furiosamente, verde, ideas, es, juan, viejo, juega, tranquilamente, duermen, duerme, cuadrado, cuadrada, el, la.

Axioma: S

Producciones:

S ⟶	NP VP	AP ⟶	Adj N
NP ⟶	N	VP ⟶	V
NP ⟶	Art AP	VP ⟶	V NP

NP	⟶ Adj AP	VP	⟶ V Pred
AP	⟶ Adj AP	Pred	⟶ Adv
N	⟶ pelota	Adj	⟶ verde
N	⟶ ideas	Adj	⟶ viejo
N	⟶ Juan	Adj	⟶ cuadrado
V	⟶ es	Adv	⟶ furiosamente
V	⟶ juega	Adv	⟶ tranquila-mente
V	⟶ duermen	Art	⟶ un
V	⟶ duerme	Art	⟶ una
Adj	⟶ incoloro	Art	⟶ el
Adj	⟶ incolora	Art	⟶ la

Nótese que este lenguaje podía haberse descrito en forma más-compacta, usando la Forma Normal Backus.

Una construcción típica ("prueba") en este lenguaje es la sucesión siguiente de sucesiones de símbolos: S, NP, VP, N VP,- N V NP, N V N, Juan V N, Juan juega N, Juan juega pelota.

Otras sentencias que pueden construirse en este lenguaje son, por ejemplo:

"Juan duerme tranquilamente"

"El viejo Juan es cuadrado"

Pero es claro que este lenguaje no es utilizable para una conversación en español, ya que en él serían permisibles expresiones tales como:

"la pelota cuadrada juega ideas".

"una idea duerme la pelota"

"la idea verde duerme furiosamente" (esta última expresión se la debemos a Chomsky).

Uno de los problemas primarios en el diseño de cualquier algoritmo de traducción mecánica es el del manejo de referencias-contextuales. En los distintos lenguajes naturales el significado de una determinada palabra depende a menudo del contexto en el que se presenta; este contexto puede extenderse incluso más allá de los límites de la frase particular en que la palabra se presenta.

Por ejemplo:

> la palabra "medida" tiene significados muy distintos en la expresión "una reacción medida", según que el tema de que se esté hablando pertenezca a la química o a la política internacional.

Esto ha motivado un estudio y clasificación de los lenguajes-formales que se basa en varias condiciones contextuales.

Por ejemplo:

> un LENGUAJE INDEPENDIENTE DEL CONTEXTO es un lenguaje en que todas las producciones tienen la forma s $\longrightarrow$ ...,- donde s es un símbolo especificado.
>
> Es decir, la producción depende solamente de símbolos específicos, y no del contexto en que aparecen. Así, el lenguaje del ejemplo 2.3 es independiente del contexto.

Hemos dicho anteriormente que las máquinas de Turing, los algoritmos de Markov y otros dispositivos análogos son descriptibles en términos de lenguajes formales.

Lo mismo es cierto para los distintos tipos de autómata que hemos estudiado.

Recíprocamente, se ha demostrado que para varias lenguas se pueden construir autómatas de diferentes tipos que reconoce--

rán o aceptarán aquellas y sólo aquellas sucesiones que sean proposiciones en el lenguaje. Además, estos autómatas tienen una clasificación natural que es paralela a la de los lenguajes formales.

Como ejemplo final de la sección damos la descripción del lenguaje del ejemplo 2.3, usando proposiciones en Forma Normal - Backus.

Ejemplo 2.3 (bis)

Descripción en Forma Normal Backus.

⟨Art⟩ ::= un | una | el | la

⟨Ad⟩ ::= incoloro | incolora | verde | viejo | cuadrado | cuadrada

⟨N⟩ ::= pelota | ideas | juan

⟨V⟩ ::= es | juega | duermen | duerme

⟨Adv⟩ ::= furiosamente | tranquilamente

⟨AP⟩ ::= ⟨Adj⟩ ⟨N⟩ | ⟨Adj⟩ ⟨AP⟩

⟨NP⟩ ::= ⟨Art⟩ ⟨AP⟩ | ⟨Adj⟩ ⟨AP⟩ | ⟨N⟩

⟨Pred⟩ ::= ⟨Adv⟩

⟨VP⟩ ::= ⟨V⟩ | ⟨V⟩ ⟨Pred⟩ | ⟨V⟩ ⟨NP⟩

⟨S⟩ ::= ⟨NP⟩ ⟨VP⟩

CAPITULO V

C O N C L U S I O N E S.

Se han examinado en este trabajo algunas de las ideas básicas que se encuentran detrás de las formas con que hoy en día la gente, con los medios de que dispone, resuelve problemas. El marco es matemático por dos razones — que se han comprendido muy bien los procesos resolutivos cuando los problemas eran los de las matemáticas clásicas, y que las matemáticas modernas expanden sus aplicaciones en áreas de las ciencias y las humanidades que tradicionalmente se han considerado no matemáticas y de naturaleza cualitativa.

Ha habido continuos comentarios, tanto en la prensa popular como en la científica, sobre el impacto de las computadoras en nuestra sociedad, que han cubierto la gama desde la automatización hasta la inteligencia artificial. Esta situación refleja lo que ha realizado el hombre en su antigua preocupación por la lógica, los algoritmos y los lenguajes, aunque este interés no ha sido siempre matemático. Por ejemplo, la necesidad de traducir lenguas surge por vez primera cuando dos grupos sociales con idiomas diferentes entran en contacto. -- Sin embargo, hasta tiempos muy recientes la traducción de una lengua a otra y, en general, los estudios lingüísticos habían seguido un curso completamente ajeno a las matemáticas.

La importancia de tener algoritmos o métodos de cálculo era ya conocida por los egipcios y los griegos de la antiguedad.- El mismo nombre de "algoritmo" se deriva de Al-Khova-razmi, el nombre de un matemático árabe (780 al 850) que escribió un tratado de álgebra aproximadamente en el año 830. El uso del término para denotar cálculos finitos en general (como opuesto a los cálculos con numerales arábicos), como aquí lo hemos usado, es relativamente reciente. El proceso tecnológico de solución de un problema, por complejo que sea, se compone de una multitud de operaciones sencillas, de pasos elementales -

sencillos.

Actualmente, con los notables progresos de la computación como rama matemática, la palabra "algoritmo" ha adquirido un sentido extraordinariamente amplio. Es una indicación exacta sobre el orden de solución de toda clase de problemas complejos mediante una serie de operaciones sencillas. En otras palabras, el algoritmo es una guía de acción para la solución de los problemas. Se le puede incluso presentar en forma de breves órdenes, que es necesario ejecutar exactamente y sin objeciones. Entonces el algoritmo sirve como un "Lazarillo" concienzudo, muestra el camino por el cual se va a la solución del problema.

La importancia de estos "guías" es tal que resulta que las matemáticas no pueden pasar sin ellos y, en nuestro caso, se puede incluso señalar que lo principal en la matemática de cómputo es la determinación de los algoritmos. Los especialistas dicen que, en la actualidad, cuando la matemática mecanizada se desarrolla rápidamente y las computadoras electrónicas se convierten en un instrumento común y corriente, aparece, cada vez con mayor frecuencia, la necesidad de encontrar el algoritmo que permita resolver tal o cual problema, o - serie de problemas. Si está encontrado el algoritmo, es posible elaborar un programa, en cuya base la máquina resolverá cualquier problema de la serie. Dado que la máquina calcula rápidamente y lo hará con mayor rapidez aún, el encontrar un algoritmo para la solución de un grupo determinado de problemas tiene una importancia máxima. Así que ES MAS VENTAJOSO ENCONTRAR UN METODO GENERAL DE SOLUCION PARA UN GRAN NUMERO DE PROBLEMAS SIMILARES Y ENCOMENDAR EL PROPIO PROCESO DE SOLUCION A LA MAQUINA, QUE BUSCAR PARA CADA PROBLEMA SU METODO Y, LUEGO, RESOLVERLO CON LA AYUDA DE LA MAQUINA O SIN ELLA.

También se trató en este trabajo el papel que ha jugado la ló

gica matemática en su aplicación al cálculo (programado) he-- cho por computadoras desde la construcción de circuitos con - interruptores como aplicación directa del álgebra Booleana. - Aunque sería absurdo en la realidad introducir en las máqui-- nas electrónicas de acción rápida circuitos eléctricos con in terruptores, estos nos ilustran claramente la manera en que - "físicamente" se interpretan los abstractos conectivos lógi-- cos de la Lógica Matemática. Para poder concluir nuestro ob- jetivo final, veamos el siguiente ejemplo de válvulas electró nicas. Hacemos notar que, fuera del ejemplo, en nuestro ca-- so, actúan dispositivos electrónicos que reproducen esas mis- mas operaciones lógicas, pero a enormes velocidades.

Nuestro ejemplo consiste de tres circuitos de válvulas elec-- trónicas:

- el primero, NO;
- el segundo, Y;
- el tercero, O.

Para explicar su funcionamiento hagamos una analogía entre la corriente eléctrica y la corriente de agua.

- Si es NO, entonces se corre el pasador y desciende el ni- vel del agua en el recipiente.

- Si se corren los pasadores Y primero, Y segundo, el nivel del agua en el recipiente desciende.

- Si se corren los pasadores O primero, O segundo, el nivel del agua desciende.

Esto se muestra en la figura V.1.

Ahora bien, en los tubos electrónicos ocurre, aproximadamente, lo mismo. Sólo que lo que fluye no es agua sino corriente --

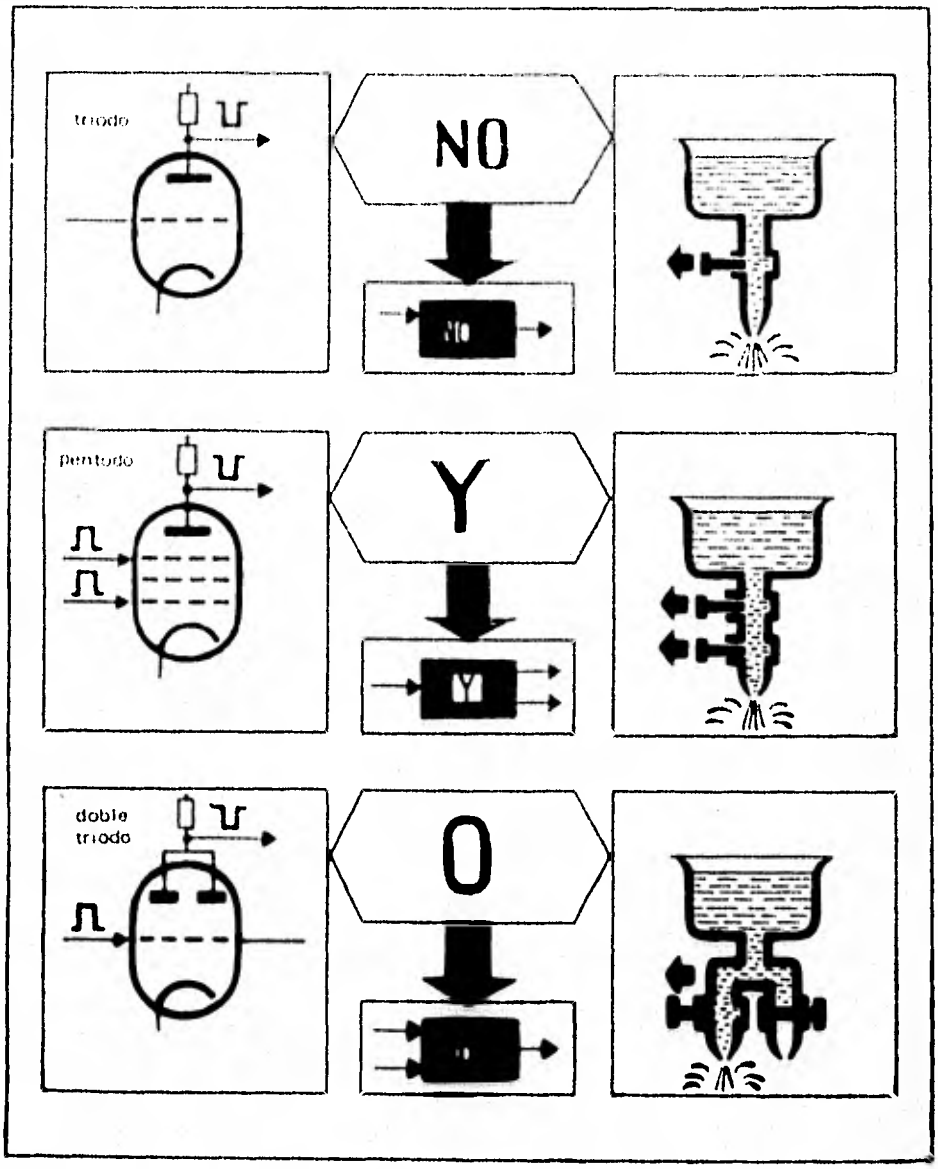


Figura V.1.

eléctrica y en lugar del nivel del agua tenemos las magnitu-- des de tensiones en la salida. Los circuitos se dirigen no - por pasadores sino por impulsos de corriente.

De esta manera se lograron los circuitos materializados en la electrónica, que realizan tres operaciones lógicas fundamentales: Y, O, NO. Son ellos los que ayudan a la máquina a "razonar" durante el funcionamiento:

Y a unir,
O a seleccionar,
No a negar.

Recordando las asignaciones del uno "1" y el cero "0", éstas- parecen decir "sí" o "no", entonces se ve claramente que el "lenguaje" de las máquinas, a pesar de toda su sencillez, permite codificar todos los números y palabras y realizar razonamientos lógicos.

La aplicación del cálculo de los enunciados en los autómatas- y en los circuitos electrónicos es lo que une la Lógica Matemática con la cibernética, obliga a esta ciencia "exclusiva - de gabinete", ciencia abstracta,a aportar resultados visibles en la práctica, ya que la Lógica matemática activa en todas - partes donde hay computadoras electrónicas. En efecto, cada- problema se resuelve en las máquinas conforme a sus leyes inmutables. Además, los propios circuitos de las máquinas calculadoras y sus redes electrónicas se analizan y elaboran con la ayuda de la Lógica Matemática.

De esta forma, la Lógica Matemática, con el desarrollo de las máquinas calculadoras, está estrechamente vinculada con la matemática de cálculo y con todo lo referente a la construcción y programación de computadoras electrónicas. Habiendo comenzado todo esto, claro, en el momento en que el hombre vió la-

posibilidad de construir circuitos eléctricos a base de la Lógica Matemática. Esto se llevó a cabo y ahora circuitos electrónicos de múltiples formas son la base de las máquinas calculadoras. Notemos que el inicio de todo esto nació en la rama que llamamos Algebra Booleana. Pues bien, aparte del uso de ésta en el diseño de las computadoras, y de los conceptos lógicos en el desarrollo de los algoritmos, la lógica misma ha entrado en dos vías de desarrollo en la computación. Aunque no hay ningún algoritmo que determine si una fórmula dada, digamos, del cálculo de predicados de primer orden (tema que ya no tratamos en este trabajo, pero sirve el comentario para los iniciados) es o no un teorema, existen, sin embargo, algoritmos para decidir esta cuestión para clases limitadas de tales fórmulas. Recientemente se han estado usando computadoras en el desarrollo de estos algoritmos y también en el desarrollo de semialgoritmos eficientes para otras clases de fórmulas. De mayor significación en la evolución de las computadoras es, probablemente, el campo de la demostración de Teoremas. Para ello, usualmente, se le proporcionan a la computadora una axiomatización del cálculo proposicional o de la geometría, algunas reglas de inferencia, y algunas guías más o menos intuitivas, aparte de un programa adecuado. Con esta base, se supone que la computadora prueba teoremas. Los ensayos en este aspecto han sido alentadores, pero no espectaculares. Sin embargo, estos intentos se han combinado en una de las fuentes del campo de la inteligencia artificial — el esfuerzo por hacer que la computadora "piense" como una persona.

Para finalizar, la evolución de las computadoras ha sido tan rápida que es tonto arriesgar una opinión sobre su evolución dentro de los próximos diez o veinte años. Pero los problemas de comunicación con la computadora y con otras personas (en parte a través de una computadora) seguirán con nosotros. Si estos y otros problemas han de resolverse con la ayuda de-

una computadora, parece que la única herramienta básica que - tenemos a nuestra disposición es la lógica matemática y la -- teoría de los algoritmos, en cualquiera de sus varias formas.

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

- Awad, Elias M.: "Procesamiento Automático de Datos": Edit.- Diana. México, 1976.

- Burks, A.W., D.W., Warren, and J.B. Wright. "An analysis- of a logical machine using parenthesis-free notacion", Mathematical Tables and other Aids to Computation, Vol. 8 ,- No. 46, abril 1954, 53-57. U.S.A.

- "Encyclopedic Dictionary of Mathematics by the Mathemati-- cal Society of Japan": The MIT Press. Cambridge, Massachusetts, 1980.

- Forsythe, Keenan, Organick, Stenberg. "Computer Science, A First Course". Edit. John Wiley & Sons, Inc. U.S.A., 1969.

- Gordon, B. Davis."Introduction to Electronic Computers." - Edit. Mc Graw- Hill, Inc. U.S.A., 1980.

- I.B.M. de México. Manuales:
  .) Fundamentos de los Ordenadores
  ..) Técnicas de Diagramación
  Madrid, 1970.

- Kleene, S.C."Mathematical Logic," John Wiley & Sons, Inc. - U.S.A., 1967.

- Kneale, W."The Development of Logic." Clarendon Press, Oxford, 1962.

- Korfhage, Robert R." Logic and Algoritms". John Wiley & - Sons., Inc., 1967.

- Kuratowski, K. "Introducción a la Teoría de Conjuntos y a la Topología." Edit. Vinces-Vives. Barcelona, 1973.

- Mendelson, E. "Introduction to Mathematical Logic." Van Nostrand, 1979.

- Multi-User Basic (Instruction Manual) Cromemco, Inc., Mountain View, California, 1979.

- Pékelis, V. "Pequeña Enciclopedia de la Gran Cibernética." Edit. MIR, Moscú, 1977.

- Rabuffetti, Hebet. "Conceptos Básicos de Matemática Moderna." Edit. Codex, Buenos Aires, 1966.

- Rado, T., S. Lin. "Computer Studies of Turing Machine Problems", Journal of the Association for Computing Machinery, No. 12, 1965. 196 - 212. U.S.A.

- Tremblay, Manohar. "Discrete Mathematical Structures with applications to Computer Science". Edit. McGraw Hill, 1975.

- Velarde, V. Carlos. "Un ejemplo de Aplicación de la Lógica Matemática Elemental a la Computación." IIMAS, U.N.A.M., México.