Diánoia, vol. 12, no. 12, 1966

UNA MECANIZACIÓN DEL CÁLCULO PROPOSICIONAL

1. *Las máquinas lógicas*

Como es sabido, Raimundo Lull o Lulio fue el inventor del primer dispositivo para combinar conceptos de un modo mecánico, o sea, de la primera máquina lógica capaz de simular razonamientos.[1] Cuatro siglos después, Leibniz advirtió la posibilidad de aplicar el método de Lulio a la lógica formal y estableció un sistema para ejecutar operaciones lógicas, empleando un código de 27 términos primitivos, representados simbólicamente por números enteros.[2] A fines del siglo XVIII, Charles Stanhope inventó su "demostrador", máquina lógica con la cual se podían ejecutar los silogismos tradicionales, realizar inferencias con cuantificadores numéricos y resolver algunos problemas elementales de cálculo de las probabilidades.[3] En 1869, William Stanley Jevons construyó su "piano lógico", la primera máquina que permitió la ejecución automática de inferencias formales, demostró experimentalmente la naturaleza del análisis lógico y suministró una prueba convincente de la superioridad de la lógica booleana respecto a la aristotélica.[4] En 1881, Allan Marquand construyó otra máquina lógica mucho más simple y práctica que la de Jevons y, además, estableció el diseño de los circuitos eléctricos necesarios para que su máquina pudiera ser operada electromagnéticamente.[5] En 1903, Annibale Pastore construyó una máquina lógica que realiza razonamientos silogísticos mediante movimientos mecánicos, o sea, que funciona de modo similar a una computadora analógica.[6]

La primera máquina lógica eléctrica fue construida por Benjamin Burack en 1936 y sirvió para comprobar experimentalmente las inferencias mediatas, incluyendo las formas condicionales y disyuntivas, lo mismo que la conversión de las proposiciones.[7] Con el descubrimiento de la estrecha analogía existente

[1] El dispositivo de Lulio (1235-1315) se encuentra descrito como parte de su *Ars Magna;* puede verse *Raymundi Lullii Opera ea quae ad adinventam ab ipso artem universalem... pertinent,* Estrasburgo, 1617. Una relación histórica detallada sobre las máquinas lógicas se encuentra en la obra de Martin Gardner, *Logic Machines and Diagrams,* Nueva York, McGraw-Hill, 1958.

[2] *Dissertio de arte combinatoria,* Leipzig, 1666.

[3] Véase, Robert Harley, "The Stanhope Demonstrador", *Mind,* vol. 4, abril de 1879.

[4] *The Substitution of Similars,* Londres, 1869.

[5] "A New Logical Machine", *Proceedings of the American Academy of Arts and Sciences,* vol. 21, 1885, p. 303.

[6] *Logica Formale, dedotta della considerazione di modelli meccanici,* Turín, 1906.

[7] "An Electrical Logic Machine", *Science,* vol. 109, 17 de junio de 1949, p. 610.

entre la teoría de las redes eléctricas y el cálculo proposicional,[8] se tuvieron las bases para construir nuevas máquinas lógicas, como la de Kalin-Burkhart,[9] en las que se establece una red eléctrica cuya trayectoria es isomorfa a la estructura lógica de las operaciones que se plantean. Así, en principio, se encuentra abierta la posibilidad de construir computadoras específicamente lógicas, que permitan explorar el inmenso dominio de las lógicas polivalentes y sirvan para experimentar nuevas operaciones, incluyendo entre ellas las que se requieren para establecer la teoría matemática de la dialéctica y para realizar los cálculos correspondientes. Pero, debido a que todavía no se han encontrado aplicaciones prácticas importantes de esas operaciones, aparentemente extrañas y complejas, no ha habido suficientes incentivos para hacer los desarrollos teóricos indispensables y construir las costosas computadoras requeridas. Sin embargo, mientras tanto, afortunadamente es posible utilizar las computadoras digitales que se encuentran en funcionamiento —todas ellas construidas para ejecutar principalmente cálculos numéricos—, de tal modo que realicen operaciones de lógica proposicional. Y, justamente, el propósito de este artículo es el de presentar un programa de cálculo proposicional formulado para ser ejecutado en una computadora digital.

2. *El lenguaje LISP*

El lenguaje que utilizaremos es el LISP, un lenguaje formal creado para ejecutar operaciones de cálculo con expresiones simbólicas. Este lenguaje se caracteriza por tener una notación muy simple, por hacer uso de símbolos atómicos, listas y combinaciones de átomos y listas para representar sus estructuras, y por emplear fundamentalmente funciones recursivas. Tales características le prestan una gran flexibilidad, tal como se ha puesto de manifiesto en las aplicaciones que ya se han hecho del LISP en la teoría de los circuitos eléctricos, el cálculo diferencial e integral, la teoría de los juegos y la teoría de la demostración. El LISP fue formulado originalmente por John McCarthy y ha sido desarrollado después por su autor y otros investigadores en el Centro de Cálculo del Massachusetts Institute of Technology, y por Harold V. McIntosh, Tomás A. Brody y otros colaboradores en el Centro Nacional de Cálculo del Instituto Politécnico Nacional.[10] En lo que sigue, presentamos el LISP especialmente adaptado como instrumento del cálculo lógico.

[8] La analogía fue sugerida por P. Ehrenfest en 1910; fue elaborada en detalle por V. I. Chestakov en 1934-1935; también fue formulada independientemente por A. Nakasima y M. Hanzawa en 1936; y quedó establecida definitivamente por C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits", *Transactions of the American Institute of Electrical Engineers*, vol. 57, diciembre de 1938, pág. 713.

[9] Véase E. C. Berkeley, "The Relations between Symbolic Logic and Large-scale calculating Machines", *Science*, vol. 112, 6 de octubre de 1950, pág. 395.

[10] J. McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation by Machine", *Communications of the ACM*, abril de 1960; McCarthy *et al.*,

Un átomo es una hilera cerrada de uno o más símbolos (letras, números u otros signos) como los siguientes:

A
WKY
AXY2Z*$L
CALCULOLOGICOENLISP
QWER+FGHJ*9704/VCXZPOUY3518*NM

que son considerados como expresiones indivisibles, de tal manera que no son analizados en sus características individuales, ni sufren alteración alguna cuando se opera con ellos. Por lo tanto, los átomos:

A B AB ACB CA

son enteramente diferentes y entre ellos sólo existirán las relaciones que se establezcan expresamente. Cada átomo se separa de los otros átomos y de las listas por medio de un espacio en blanco; aunque es indiferente si en lugar de uno se emplean varios espacios, cuando así se estime conveniente para dar mayor claridad a la expresión.

Una lista es un conjunto ordenado de elementos, cuya sucesión se escribe entre paréntesis. Dentro de la lista, cada elemento se separa del antecesor y del sucesor por medio de uno o más espacios, aun cuando el primer elemento se puede escribir inmediatamente después del paréntesis izquierdo, y el último inmediatamente antes del paréntesis derecho. Cuando todos los elementos de una lista son átomos, se tiene una lista linear, como sucede en los siguientes ejemplos:

(A B CD R2P F H)
(M S AXY6 CA N9)
(ESTA ES UNA LISTA EN LISP)

También pueden figurar como elementos de una lista otras listas, en cuyo caso la lista en cuestión es no-linear, como ocurre con las listas siguientes:

(R M (A B CD) S (T)), que consta de 5 elementos;
(F P4 R*$ (K) T1 H (CGR Y) W), que tiene 8 elementos; y
(K (& (+ (MS TYU (X) HGKJDE) *6N))), que consta de 2 elementos.

LISP 1.5 Programmer's Manual, The Massachusetts Institute of Technology Press, Cambridge, Mass., 17 de agosto de 1962; H. V. McIntosh, *et al., LISP Conversion,* Program Note No. 3, México, C.I.E.A. y Centro Nacional de Cálculo del I.P.N., 1965; T. A. Brody, *LISPITO, a LISP Processor for the IBM 1620* Centro Nacional de Cálculo del I.P.N., México, 1965; Brody, *Symbol Manipulation Techniques for Physics,* Preliminary Version (se publicará en 1966).

Un caso particular es el de la lista vacía o nula, que no contiene ningún elemento y se representa, por lo tanto, así:

( )

Sin embargo, es necesario no confundir la lista nula con la lista que tiene como único elemento a la lista nula, cuya expresión es:

(())

Por lo tanto, es fácil advertir que los únicos símbolos que tienen asignada una función específica en el LISP, desde un principio, son los dos paréntesis y los espacios en blanco.

3. *Funciones recursivas*

Una función recursiva es aquella que se define mediante la propia función, solo que referida a diferentes argumentos. Por ejemplo, una definición recursiva de la potencia enésima de un número es la siguiente:

$$x^n = \begin{cases} 1, & \text{cuando: } n = 0 \\ x \cdot x^{n-1}, & \text{cuando: } n \neq 0 \end{cases}$$

En efecto, para el valor: $n = 3$, tendremos:

$$\begin{aligned} x^3 &= x \cdot x^{3-1} = \\ &= x \cdot x^2 = x \cdot x \cdot x^{2-1} = \\ &= x \cdot x \cdot x^1 = x \cdot x \cdot x \cdot x^{1-1} = \\ &= x \cdot x \cdot x \cdot x^0 = x \cdot x \cdot x \cdot 1 = \\ &= x \cdot x \cdot x \end{aligned}$$

Como se puede observar, para que la ejecución de una definición recursiva no resulte interminable, es necesario hacer que por lo menos uno de los valores de la función no quede definido recursivamente. Esa parte no-recursiva de la definición es una condición terminal. En el caso del ejemplo anterior, la condición terminal es:

$$x^n = 1, \quad \text{cuando } n = 0.$$

Por supuesto, una definición recursiva puede ser mucho más complicada que la del ejemplo, ya que puede referirse a un mayor número de argumentos, tener varias condiciones terminales, incluir como partes integrantes a otras funciones recursivas y, más aún, puede darse el caso de que una función recursiva empleada en la definición de otra, incluya en su propia definición a la función principal que se esté definiendo.

En LISP, las definiciones se formulan generalmente utilizando las funciones recursivas *lambda,*[11] que se caracterizan por asociar de un modo inequívoco los valores que se den a las variables. Además, lo que es de mayor importancia aún, se ha demostrado que toda función lambda es computable. Por ejemplo, una función definida ordinariamente por:

$$f(x, y) = y + x^2,$$

se expresa como función lambda de la siguiente manera:

$$(f\ (\text{lambda}\ (x\ y)\ y + x^2)).$$

Las variables se encuentran así ligadas por la lambda y, en consecuencia, al aplicar la función a un caso concreto, los valores de los argumentos deben ser tomados en el mismo orden en que están ligadas las variables. Por lo tanto, en el ejemplo, su valor queda bien definido cuando los valores de los argumentos son 3 y 4, ya que entonces:

$$(f\ (\text{lambda}\ (x\ y)\ y + x^2)\ 3\ 4) = 4 + 3^2 = 13;$$

lo mismo que en el caso de que los valores sean 4 y 3, pues así:

$$(f\ (\text{lambda}\ (x\ y)\ y + x^2)\ 4\ 3) = 3 + 4^2 = 19.$$

Por otra parte, las variables ligadas pueden ser sustituidas sin que se altere la función, siempre que la sustitución se haga en forma consecuente. Por consiguiente, las tres expresiones que siguen representan la misma función:

$$(f\ (\text{lambda}\ (x\ y)\ y + x^2))$$
$$(f\ (\text{lambda}\ (r\ s)\ s + r^2))$$
$$(f\ (\text{lambda}\ (s\ r)\ r + s^2)).$$

También puede haber variables libres, que no estén ligadas por la lambda como, por ejemplo, en la función:

$$(f\ (\text{lambda}\ (x\ y)\ x^n + y^m)),$$

la *n* y la *m* son variables libres, que pueden ser consideradas como parámetros. Por supuesto, cuando en la definición de una función aparecen variables libres, es indispensable darles un valor determinado antes de ejecutar operaciones, para que el valor de la función quede definido.

[11] Alonzo Church, *The Calculi of Lambda-Conversion,* Princeton, Princeton University Press, 1941.

4. *Funciones primitivas*

Para los propósitos de este artículo emplearemos como operaciones primitivas tres funciones, que se conocen con los nombres de CAR, CDR y CONS; tres predicados, ATOM, EQ y NULL; y dos seudo-funciones, QUOTE e IF.[12] (En adelante usaremos letras mayúsculas en las expresiones, debido a que en las computadoras no se emplean minúsculas.)

La función CAR se aplica a un solo argumento, el cual tiene que ser una lista. La operación que realiza consiste en evaluar su argumento y extraer de esa lista el primer elemento, que puede ser un átomo o una lista. La función CAR no se encuentra definida para el caso de que el argumento sea un átomo, ni tampoco cuando se trate de una lista nula. Ejemplos:

(CAR (A B C D)) = A
(CAR (X (Y M) R)) = X
(CAR ((S L) F H)) = (S L)
(CAR ((M (R)) T E)) = (M (R))
(CAR (J)) = J
(CAR (( ))) = ( )
(CAR X), si X es un átomo, no está definida
(CAR ( )), no está definida.

La función CDR se aplica a un solo argumento, el cual tiene que ser necesariamente una lista. La operación que realiza es la de evaluar su argumento y suprimir de la lista su primer elemento, ya sea un átomo o una lista, formando una nueva lista sin ese elemento. Cuando el argumento tiene un solo elemento, entonces el resultado de esta operación es una lista nula. La función CDR no está definida para el caso de que el argumento sea un átomo, ni tampoco cuando se trata de una lista nula. Ejemplos:

(CDR (R M S)) = (M S)
(CDR (A (X Z))) = ((X Z))
(CDR ((A B) M)) = (M)
(CDR (L)) = ( )
(CDR ((Y O))) = ( )
(CDR (( ))) = ( )
(CDR X), si X es un átomo, no está definida
(CDR ( )), no está definida.

[12] *LISP 1:5 Programmer's Manual;* J. McCarthy, *A Basis for a Mathematical Theory of Computation,* en *Computer Programming and Formal Systems,* ed. P. Braffort y D. Hirschberg, Amsterdam, North-Holland, 1963, págs. 33-70.

La función CONS sirve para constituir listas y se aplica a dos argumentos, el primero de los cuales puede ser un átomo o una lista y el segundo tiene que ser necesariamente una lista, inclusive una lista nula. La operación que ejecuta es la de introducir el primer argumento como primer elemento de la lista que constituye su segundo argumento. Ejemplos:

(CONS S (A B CD)) = (S A B CD)
(CONS (T WE) (MN B G)) = ((T WE) MN B G)
(CONS R ( )) = (R)
(CONS ( ) (X Y ZC)) = (( ) X Y ZC)
(CONS ( ) ( )) = (( ))
(CONS X Y), si Y es un átomo, no está definida.

Las funciones CAR y CDR se pueden repetir y combinar de diferentes maneras, formando así funciones compuestas, como las siguientes:

(CAR (CDR (A B C))) = (CAR (B C)) = B
(CAR (CAR ((M R) S T))) = (CAR (M R)) = M
(CDR (CAR ((M R) S T))) = (CDR (M R)) = (R)
(CDR (CDR ((M R) S T))) = (CDR (S T)) = (T)
(CDR (CDR (CAR ((M R) S T)))) =
= (CDR (CDR (M R))) = (CDR (R)) = ( )

En tales casos, los nombres de las funciones resultantes se pueden componer del siguiente modo:

(CAR (CAR X)) = (CAAR X)
(CAR (CDR X)) = (CADR X)
(CDR (CAR X)) = (CDAR X)
(CAR (CAR (CDR (CAR (CDR X))))) = (CAADADR X)

Naturalmente, tales funciones compuestas estarán definidas únicamente cuando los argumentos cumplan las condiciones pertinentes. Una función notable de este tipo es la que permite obtener el elemento de orden *n* de una lista, que es:

(CAR X) = primer elemento de X
(CADR X) = segundo elemento de X
(CADDR X) = tercer elemento de X
. . . . . .
(CADD...DR X) = enésimo elemento de X, cuando las D correspondientes a los sucesivos CDR se repiten (n — 1) veces.

También la función CONS se puede reiterar todas las veces que se necesite, o bien, combinar con CAR y CDR. Ejemplos:

(CONS X (CONS Y (Z))) = (X Y Z)
(CONS A (CONS B (CONS C (CONS D ( ))))) = (A B C D)
(CONS M (CONS (N) (R))) = (M (N) R)
(CONS (CAR (A R O)) (CDR (X Y Z))) = (A Y Z)
(CONS (CDR (A R O)) (CAR ((S) Y Z))) = ((R O) S)

Entre las combinaciones posibles, resultan las siguientes equivalencias notables, que se cumplen para dos listas L y M cualesquiera:

(CONS (CAR L) (CDR L)) = L
(CAR (CONS L M)) = L
(CDR (CONS L M)) = M.

Los predicados son funciones cuyo espacio o universo está limitado a los valores *T* y *F*, para la verdad y la falsedad, respectivamente. Como ya lo dijimos antes, los predicados primitivos que utilizaremos son ATOM, EQ y NULL.

El predicado ATOM se aplica a un solo argumento. Su valor es T *sii* (es decir, si y sólo si) su argumento es un átomo; en caso contrario, su valor es F. Por lo tanto:

$$(\text{ATOM X}) = \begin{cases} \text{T, si X es un átomo} \\ \text{F, en cualquier otro caso} \end{cases}$$

Ejemplos:

(ATOM PRGKJH) = T
(ATOM L) = F, si: L = (M R S)
(ATOM M) = F, si: M = (A)
(ATOM (CAR (H I J))) = T
(ATOM (CDR (H I J))) = F.

El predicado EQ se aplica a dos argumentos. Su valor es T *sii* los dos argumentos son átomos y éstos son iguales; en caso contrario, su valor es F. De aquí que:

$$(\text{EQ X Y}) = \begin{cases} \text{T, si: X y Y son átomos y, además, X} = \text{Y} \\ \text{F, en cualquier otro caso} \end{cases}$$

Ejemplos:

(EQ B B) = T
(EQ AM MA) = F
(EQ R (CAR (R S P))) = T
(EQ S (S)) = F

El predicado NULL se aplica a un solo argumento. Su valor es T *sii* el argumento es una lista nula; en caso contrario, su valor es F. Así, tenemos:

$$(\text{NULL X}) = \begin{cases} \text{T, si: X} = (\ ) \\ \text{F, en cualquier otro caso} \end{cases}$$

Ejemplos:

(NULL ( )) = T
(NULL (R)) = F
(NULL (CDR (R))) = T
(NULL (( ))) =F
(NULL R) = F.

Las seudo-funciones expresan operaciones en las cuales no se evalúan todos los argumentos. Como hemos dicho, utilizaremos aquí las seudo-funciones QUOTE e IF.

La seudo-función QUOTE se aplica a un solo argumento. La operación que ejecuta consiste en la evaluación de su argumento, tomando siempre como valor el *nombre* de su argumento. Por consiguiente, esta seudo-función es la que permite distinguir en LISP entre una expresión cualquiera X y su nombre "X". Ejemplos:

(QUOTE X) = X
(QUOTE (X)) = (X)
(QUOTE P1R2Z) = P1R2Z
(QUOTE (CADR X)) = (CADR X)
(QUOTE ( )) = ( )
(QUOTE (M NO T)) = (M NO T).

La seudo-función IF se aplica a tres argumentos. El primer argumento tiene que ser necesariamente un predicado y los otros dos pueden ser expresiones de cualquier tipo, inclusive también predicados. La operación que realiza es la de evaluar el primer argumento y, si el valor de éste es T, entonces evalúa el segundo argumento y su valor es el de toda la expresión a la que se esté aplicando la seudo-función IF. Cuando el valor del primer argumento es F, entonces evalúa el tercer argumento y su valor es el que corresponde a la expresión entera. Por lo tanto, tenemos:

$$(\text{IF X Y Z}) = \begin{cases} \text{Y, si: X} = \text{T} \\ \text{Z, si: X} = \text{F} \end{cases}$$

Además, la seudo-función IF puede tener argumentos compuestos, incluso formados por otro u otros IF. Ejemplos:

Si: X es un átomo, L = (A M R), Y = (G O R), Z = ( ), W = (S), entonces:

(IF (ATOM X) (CAR L) (CDR L)) = A
(IF (ATOM Y) (CAR L) (CDR L)) = (M R)
(IF (NULL Z) L (CONS Z L)) = (A M R)
(IF (NULL W) L (CONS W L)) = ((S) A M R)
(IF (ATOM W) (CAR L) (IF (NULL W) (CDR L)
(CONS (CAR W) L))) = (S A M R)

Por último, vamos a referirnos a otros tres predicados, AND, OR y NOT, aunque únicamente los utilizaremos aquí para definir las funciones proposicionales y para indicar los valores *T* y *F*, tal como lo señalamos a continuación.

El predicado AND se aplica a cualquier número de argumentos, incluso a ninguno. Su valor es T *sii* ninguno de sus argumentos es F. Por ende, resulta que cuando no se tiene ningún argumento, el valor de AND es siempre T, o sea, que:

$$(\text{AND}) = \text{T}$$

El predicado OR se aplica también a un número cualquiera de argumentos, incluso a ninguno. Su valor es F *sii* ninguno de sus argumentos es T. En consecuencia, tenemos que cuando no se tiene ningún argumento, el valor de OR es siempre F, o sea, que:

$$(\text{OR}) = \text{F}$$

Por lo anterior, en nuestras expresiones usaremos (AND) en vez de T, y (OR) en lugar de F; con lo cual se sigue manteniendo la situación de que los únicos símbolos reservados en LISP son los paréntesis y los espacios en blanco

El predicado NOT se puede aplicar a cualquier número de argumentos, salvo a ninguno. La operación que ejecuta consiste en evaluar su primer argumento y adoptar el valor opuesto para toda la expresión. Por lo tanto, tenemos:

$$(\text{NOT A B C} \ldots \text{N}) = \begin{cases} \text{T, si: A es F} \\ \text{F, si: A es T} \end{cases}$$

5. *Definición de funciones*

En LISP, la definición de cada función se establece con base en las funciones primitivas y utilizando la conversión recursiva lambda. El esquema general de la definición es una lista de tres elementos:

(LAMBDA (X1 X2...XN) (FORMA))

El primer elemento de la lista, el átomo LAMBDA, indica que el siguiente elemento es la lista de las variables de la función, (X1 X2 ... XN), a las cuales se les asignan después los valores específicos correspondientes; y el tercer elemento, la lista (FORMA), contiene la descripción de las operaciones por ejecutar. Por supuesto, los argumentos pueden ser, a su vez, expresiones que deban ser evaluadas y cuyos valores sean asignados a las variables durante la evaluación de la forma. También puede haber en la forma otras variables que no se encuentren ligadas dentro de la expresión lambda y que, por lo tanto, sean variables libres. En tal caso, como ya lo decíamos anteriormente, es necesario fijar previamente el valor de esas variables, para que la evaluación de la forma quede determinada. Además, puede ocurrir que no se tenga ninguna variable ligada, en cuyo caso la lista de variables será una lista vacía, que siempre debe indicarse así explícitamente.

Ahora bien, para establecer las condiciones de la ejecución de las funciones, se utilizan tres funciones no-recursivas, que son DEFINE, APPLY y FINENT.

La función DEFINE se aplica a cualquier número de argumentos y cada uno de ellos es una lista formada por dos argumentos. La operación que ejecuta es la de asociar el nombre de cada función a su descripción funcional, de la siguiente manera:

```
(DEFINE (FUNCION1 (LAMBDA (X1) (FORMA 1)))
        (FUNCION2 (LAMBDA (X1 X2) (FORMA 2)))
        (FUNCION3 (LAMBDA (X3 X4 X2) (FORMA 3)))
        .....................................................
        (FUNCIONN (LAMBDA (X5 X8 XJ XK) (FORMAN)))  )
```

Como se advierte, el primer elemento de cada una de las listas es el nombre de la función, que debe ser un átomo, y el segundo elemento es la definición de la propia función. Entonces, mediante la función DEFINE se asocian los nombres de todas las funciones que van a ser empleadas en un programa determinado con sus definiciones respectivas y, luego, al establecer los argumentos a los que deban aplicarse, basta con indicar solamente el nombre de la función, como veremos más adelante. La función DEFINE también puede servir para dar otro nombre a una función ya definida, como por ejemplo, suponiendo que tenemos ya definida la función RECIPROCIDAD, podemos atribuirle otro nombre por alguna razón conveniente, como es el de EQUIVALE, de la siguiente manera:

```
(DEFINE  (EQUIVALE  RECIPROCIDAD)  )
```

La función APPLY es la que ordena la ejecución de una operación previamente definida, haciendo que la computadora opere con los argumentos específicos que se le suministran, sin evaluarlos. Esta función se expresa en forma de lista, cuyo primer elemento es el átomo constituido por su nombre, APPLY,

que ordena la ejecución; su segundo elemento es el nombre de la operación por ejecutar; y los siguientes elementos son los argumentos específicos que deben manejarse. Por lo tanto, su esquema general es:

(APPLY FUNCION X A1 A2 ... AN)

Por último, la función FINENT sirve para ordenar a la computadora que suspenda la lectura del programa e inicie la ejecución de las operaciones indicadas. Es la única función que se expresa en LISP sin el paréntesis izquierdo, o sea, así:

FINENT)

Para ejemplificar lo antes dicho, estableceremos la definición de la función IGUAL, que es un predicado que determina si dos expresiones son iguales o no; ya que el predicado primitivo EQ no está definido para el caso de que sus dos argumentos sean listas. Pues bien, si los dos argumentos, X y Y, son átomos, la condición para que la función IGUAL sea T, es que (EQ X Y) sea T. Si uno de los argumentos es una lista nula, el otro argumento tendrá que ser también una lista nula, para que el valor del predicado IGUAL sea T. En fin, si ambos argumentos son listas no nulas, entonces tendrá que cumplirse la condición de que tanto sus respectivos CAR, como sus CDR correspondientes, sean iguales. Por lo tanto, podemos formular su definición de este modo:

```
(DEFINE
  (IGUAL (LAMBDA (X Y)
    (IF (ATOM X) (EQ X Y)
      (IF (ATOM Y) (OR)
        (IF (NULL X) (NULL Y)
          (IF (NULL Y) (OR)
            (IF (IGUAL (CAR X) (CAR Y))
              (IGUAL (CDR X) (CDR Y))
                (OR))))))  ))  )
```

Como puede advertirse, en esta definición tenemos cuatro condiciones terminales y una condición recursiva. Por otra parte, así tenemos formulado ya un brevísimo programa, que solamente consta de una función. Unicamente nos hace falta establecer los casos específicos de aplicación que podían ser, por ejemplo, los siguientes:

```
(APPLY  IGUAL  (A B C) (A B C)  )
(APPLY  IGUAL  (A (B C)) ((A B) C)  )
(APPLY  IGUAL  (CAR ((L M N) O P Q)) (CDR (K L M N))  )
(APPLY  IGUAL  (R S T U V) (CONS (CAR (R S T U V))
                                 (CDR ((R) S T U V))  )  )
(APPLY  IGUAL  (A B C)  (C B A)  )
FINENT)
```

Entonces, al hacer que una computadora ejecute dicho programa, sus respuestas serían respectivamente:

T
F
T
T
F

6. *Funciones proposicionales*

Hasta ahora, las proposiciones se han expresado en LISP como funciones de los predicados AND, OR y NOT. Así, las 16 proposiciones posibles respecto a dos términos, X y Y, se pueden formular de la siguiente manera:

```
(DEFINE  (PROFASIS (LAMBDA (X Y)
                   X))
         (PROFASISINVERSA (LAMBDA (X Y)
                   Y))
         (ANTIFASIS (LAMBDA (X Y)
                   (NOT X) ))
         (ANTIFASISINVERSA (LAMBDA (X Y)
                   (NOT Y) ))
         (CONJUNCION (LAMBDA (X Y)
                   (AND X Y) ))
         (DISCORDANCIA (LAMBDA (X Y)
                   (AND X (NOT Y)) ))
         (DISCORDANCIAINVERSA (LAMBDA (X Y)
                   (AND (NOT X) Y) ))
         (HETEROFASIS (LAMBDA (X Y)
                   (AND (NOT X) (NOT Y)) ))
         (INCLUSION (LAMBDA (X Y)
                   (OR X Y) ))
         (IMPLICACION (LAMBDA (X Y)
                   (OR (NOT X) Y) ))
         (IMPLICACIONINVERSA (LAMBDA (X Y)
                   (OR X (NOT Y)) ))
         (INCOMPATIBILIDAD (LAMBDA (X Y)
                   (OR (NOT X) (NOT Y)) ))
```

```
(EXCLUSION (LAMBDA (X Y)
           (AND (OR X Y) (NOT (AND X Y)) ) ))

(RECIPROCIDAD (LAMBDA (X Y)
           (OR (AND X Y) (NOT (OR X Y)) ) ))

(TAUTOLOGIA (LAMBDA (X Y)
           (AND) ))

(ENANTIOSIS (LAMBDA (X Y)
           (OR) ))
)
```

A las anteriores definiciones podemos agregar la función EQUIVALE, que realiza la operación de evaluar sus argumentos y adopta el valor T *sii* ellos son equivalentes. Como es fácil advertir, su definición se puede expresar así, para dos, tres y cuatro argumentos:

```
(DEFINE (EQUIVALE (LAMBDA (R S)
                  (IF R S (NOT S)) ))

        (EQUIVALE* (LAMBDA (Q R S)
                  (IF Q (IF R S (OR))
                        (IF (NOT R) (NOT S) (OR)) ) ))

        (EQUIVALE** (LAMBDA (P Q R S)
                  (IF P (IF Q (IF R S (OR)) (OR))
                        (IF (NOT Q) (IF (NOT R)) (NOT S)
                              (OR)) ) ))
)
```

Pues bien, utilizando las funciones anteriores, es posible establecer todas las equivalencias correspondientes al cálculo proposicional, incluyendo las relativas a las inferencias mediatas, las disyuntivas, las dilemáticas y las inductivas amplificadoras.[13]

7. *Listas proposicionales*

Las listas proposicionales, que aquí se establecen por primera vez, nos permiten expresar las proposiciones en forma de listas. Con esta nueva manera de expresar las proposiciones, tenemos la ventaja de poderles aplicar las funciones correspondientes a las listas. Por lo tanto, se amplía enormemente su campo de

[13] Como podrá verse en el trabajo del autor *Cálculo Lógico en LISP*, en preparación.

operación, ya que así resulta posible tratar las proposiciones como conjuntos y ejecutar con ellas las operaciones respectivas.

Para formar las 48 listas proposicionales posibles respecto a tres términos, tomados de dos en dos, utilizaremos los átomos X, Y y Z para representar dichos términos, y los átomos NX, NY y NZ para representar sus negaciones respectivas.[14] Como es fácil advertir, con esos 6 átomos son posibles 8 combinaciones diferentes de 3 átomos, sin que se repita ninguno ni tampoco figure en la misma combinación un término y su negación. Tales combinaciones nos dan las 8 listas siguientes:

(X Y Z)	(X Y NZ)	(X NY Z)	(X NY NZ)
(NX Y Z)	(NX Y NZ)	(NX NY Z)	(NX NY NZ)

Entonces constituyendo ahora la lista que tenga como elementos a esas 8 listas, tendremos expresado el universo de discurso dentro del cual operaremos:

((X Y Z) (X Y NZ) (X NY Z) (X NY NZ)
(NX Y Z) (NX Y NZ) (NX NY Z) (NX NY NZ))

Y, por ende, todas las listas proposicionales que emplearemos serán subconjuntos de este conjunto universal.

Así, la conjunción entre los términos X y Y queda expresada por la lista:

((X Y Z) (X Y NZ))

apoyándonos en la propiedad de que la conjunción entre dos términos es equivalente a la disyunción de esos dos términos y un tercero, y la conjución entre los mismos dos términos y la negación del tercero, o sea:

(EQUIVALE (AND X Y)
(OR (AND X Y Z) (AND X Y (NOT Z))))

Dicho de otro modo, tenemos que la lista:

((X Y Z) (X Y NZ))

expresa la conjunción entre X y Y, en un sentido análogo al que se utiliza para obtener la derivada parcial de una función, ya que solamente se consideran como variables a X y Y, mientras que Z y NZ son tratados como parámetros constantes. De manera similar, la conjunción entre Y y Z queda expresada por la lista:

((X Y Z) (NX Y Z))

[14] Utilizando la notación establecida por Jan Lukasiewicz en *Elementy logiki matematycznej*, Varsovia, 1929.

y la conjunción entre Z y X, por:

((X Y Z) (X NY Z))

Con base en lo antes establecido, podemos formular las definiciones de las listas proposicionales que manejaremos:

```
(DEFINE
    (UNIVERSOXYZ (LAMBDA ( )
      (QUOTE ( (X Y Z) (X Y NZ) (X NY Z) (X NY NZ)
        (NX Y Z) (NX Y NZ) (NX NY Z) (NX NY NZ) ) ) ))
    (PROFASISXY (LAMBDA ( )
      (QUOTE ( (X Y Z) (X Y NZ) (X NY Z) (X NY NZ) ) ) ))
    (PROFASISYX (LAMBDA ( )
      (QUOTE ( (X Y Z) (X Y NZ) (NX Y Z) (NX Y NZ) ) ) ))
    (ANTIFASISXY (LAMBDA ( )
      (QUOTE ((NX Y Z) (NX Y NZ) (NX NY Z) (NX NY NZ)) ) ))
    (ANTIFASISYX (LAMBDA ( )
      (QUOTE ((X NY Z) (X NY NZ) (NX NY Z) (NX NY NZ)) ) ))
    (CONJUNCIONXY (LAMBDA ( )
      (QUOTE ( (X Y Z) (X Y NZ) ) ) ))
    (DISCORDANCIAXY (LAMBDA ( )
      (QUOTE ( (X NY Z) (X NY NZ) ) ) ))
    (DISCORDANCIAYX (LAMBDA ( )
      (QUOTE ( (NX Y Z) (NX Y NZ) ) ) ))
    (HETEROFASISXY (LAMBDA ( )
      (QUOTE ( (NX NY Z) (NX NY NZ) ) ) ))
    (INCLUSIONXY (LAMBDA ( )
      (QUOTE ( (X Y Z) (X Y NZ) (X NY Z) (X NY NZ)
                (NX Y Z) (NX Y NZ) ) ) ))
    (INCOMPATIBILIDADXY (LAMBDA ( )
      (QUOTE ( (X NY Z) (X NY NZ) (NX Y Z) (NX Y NZ)
                (NX NY Z) (NX NY NZ) ) ) ))
    (IMPLICACIONXY (LAMBDA ( )
      (QUOTE ( (X Y Z) (X Y NZ) (NX Y Z) (NX Y NZ)
                (NX NY Z) (NX NY NZ) ) ) ))
    (IMPLICACIONYX (LAMBDA ( )
      (QUOTE ( (X Y Z) (X Y NZ) (X NY Z) (X NY NZ)
                (NX NY Z) (NX NY NZ) ) ) ))
    (EXCLUSIONXY (LAMBDA ( )
      (QUOTE ( (X NY Z) (X NY NZ) (NX Y Z) (NX Y NZ) ) ) ) )
    (RECIPROCIDADXY (LAMBDA ( )
      (QUOTE ( (X Y Z) (X Y NZ) (NX NY Z) (NX NY NZ) ) ) ) )
```

```
(ENANTIOSISXY (LAMBDA ( )
   (QUOTE  ( )  ) ))
(UNIVERSOYZX       UNIVERSOXYZ)
(PROFASISYZ      PROFASISYX)
(PROFASISZY (LAMBDA ( )
   (QUOTE ( (X Y NZ) (NX Y NZ) (X NY NZ) (NX NY NZ)))))
(ANTIFASISYZ       ANTIFASISYX)
(ANTIFASISZY (LAMBDA ( )
   (QUOTE ( (X Y NZ) (NX Y NZ) (X NY NZ) (NX NY NZ))))
(CONJUNCIONYZ (LAMBDA ( )
   (QUOTE  ( (X Y Z) (NX Y Z)  )  ) ))
(DISCORDANCIAYZ (LAMBDA ( )
   (QUOTE  ( (X Y NZ) (NX Y NZ)  )  ) ))
(DISCORDANCIAZY (LAMBDA ( )
   (QUOTE  ( (X NY Z) (NX NY Z)  )  ) ))
(HETEROFASISYZ (LAMBDA ( )
   (QUOTE  ( (X NY NZ) (NX NY NZ)  )  ) ))
(INCLUSIONYZ (LAMBDA ( )
   (QUOTE ( (X Y Z) (NX Y Z) (X Y NY) (NX Y NZ)
                  (X NY Z) (NX NY Z)  )  ) ))
(INCOMPATIBILIDADYZ (LAMBDA ( )
  (QUOTE ( (X NY Z) (NX NY Z) (X Y NZ) (NX Y NZ)
                 (X NY NZ) (NX NY NZ)  )  ) ))
(IMPLICACIONYZ (LAMBDA ( )
   (QUOTE ( (X Y Z) (NX Y Z) (X NY Z) (NX NY Z)
                   (X NY NZ) (NX NY NZ)  )  ) ))
(IMPLICACIONZY (LAMBDA ( )
   (QUOTE  ( (X Y Z) (NX Y Z) (X Y NZ) (NX Y NZ)
                  (X NY NZ) (NX NY NZ)  )  ) ))
(EXCLUSIONYZ (LAMBDA ( )
   (QUOTE ( (X Y NZ) (NX Y NZ) (X NY Z) (NX NY Z) ) ) ))
(RECIPROCIDADYZ (LAMBDA ( )
   (QUOTE ( (X Y Z) (NX Y Z) (X NY NZ) (NX NY NZ) ) ) ))
(ENANTIOSISYZ      ENANTIOSISXY)
(UNIVERSOZXY       UNIVERSOXYZ)
(PROFASISZX      PROFASISZY)
(PROFASISXZ      PROFASISXY)
(ANTIFASISZX      ANTIFASISZY)
(ANTIFASISXZ      ANTIFASISXY)
(CONJUNCIONZX (LAMBDA ( )
   (QUOTE  ( (X Y Z) (X NY Z)  )  ) ))
```

```
(DISCORDANCIAZX (LAMBDA ( )
   (QUOTE  (  (NX Y Z) (NX NY Z)  )  )  ))
(DISCORDANCIAXZ (LAMBDA ( )
   (QUOTE  (  (X Y NZ) (X NY NZ)  )  )  ))
(HETEROFASISZX (LAMBDA ( )
   (QUOTE  (  (NX Y NZ) (NX NY NZ)  )  )  ))
(INCLUSIONZX (LAMBDA ( )
      (QUOTE  (  (X Y Z) (X NY Z) (X Y NZ) (X NY NZ)
                 (NX Y Z) (NX NY Z)  )  )  ))
(INCOMPATIBILIDADZX (LAMBDA ( )
   (QUOTE  (  (X Y NZ) (X NY NZ) (NX Y Z) (NX NY Z)
                 (NX Y NZ) (NX NY NZ)  )  )  ))
(IMPLICACIONZX (LAMBDA ( )
   (QUOTE ( (X Y Z) (X NY Z) (X Y NZ) (X NY NZ)
                 (NX Y NZ) (NX NY NZ)  )  )  ))
(IMPLICACIONXZ (LAMBDA ( )
   (QUOTE  (  (X Y Z) (X NY Z) (NX Y Z) (NX NY Z)
                 (NX Y NZ) (NX NY NZ)  )  )  ))
(EXCLUSIONZX (LAMBDA ( )
   (QUOTE ( (X Y NZ) (X NY NZ) (NX Y Z) (NX NY Z) ) ) ))
(RECIPROCIDADZX (LAMBDA ( )
   (QUOTE ( (X Y Z) (X NY Z) (NX Y NZ) (NX NY NZ) ) ) ))
(ENANTIOSISZX        ENANTIOSISXY)
)
```

8. *Definición de operaciones*

La ejecución de las operaciones con listas proposicionales que presentaremos a continuación, requiere la definición previa de las funciones que utilizaremos.

La función INTERSECCION se aplica a dos argumentos, que deben ser listas, y realiza la operación consistente en formar una lista con los elementos que son comunes a las listas que constituyen sus argumentos.

La función UNION se aplica también a dos argumentos, que deben ser listas, y ejecuta la operación de formar otra lista constituida por los elementos pertenecientes a las dos listas que son sus argumentos.

La función DIFERENCIA se aplica igualmente a dos argumentos, que deben ser listas, y realiza la operación de constituir otra lista formada con los elementos que pertenecen a uno u otro de los argumentos, pero no a ambos.[15]

[15] Como se puede advertir, la función DIFERENCIA es considerada aquí en un sentido más amplio del que se le atribuye cuando se utiliza como análoga de la sustracción. En efecto, en este último sentido, la diferencia entre P y Q (*i.e.*, P − Q) es la lista formada por los elementos de P que no pertenecen a Q; en tanto que la diferencia entre Q y P (o sea, Q − P) es la lista constituida por los elementos de Q que no pertenecen a P: y, por lo tanto : (P − Q) ≠ (Q − P). En cambio, en el sentido en que la definimos

Las tres funciones anteriores se pueden aplicar reiteradamente, de manera que, por ejemplo, la INTERSECCION de cuatro listas P, Q, R y S es equivalente a la INTERSECCION entre la primera lista y la INTERSECCION de la segunda con la INTERSECCION de la tercera y la cuarta, o sea, que:

```
(EQUIVALE (INTERSECCION P Q R S)
          (INTERSECCION P (INTERSECCION Q
                                (INTERSECCION R S)))  )
```

La función COMPLEMENTO se aplica a un solo argumento, que debe ser una lista, y la operación que ejecuta es la de formar una lista constituida por los elementos del universo de discurso que no pertenecen al argumento, o sea, por la DIFERENCIA entre el argumento y la lista universal.

La función ELEMENTO se aplica a dos argumentos, el segundo de los cuales tiene que ser una lista, y la operación que realiza es la de determinar si el primer argumento se encuentra comprendido como elemento de la lista que constituye el segundo argumento.

La función ELIMINA se aplica a dos argumentos, el segundo de los cuales debe ser una lista, y ejecuta la operación de hacer desaparecer del segundo argumento al primero, en caso de que esté comprendido como elemento.

La función IGUAL ya la hemos definido con anterioridad y sólo recordaremos que cuando se aplica a dos listas, aún cuando éstas consten de los mismos elementos y sólo de ellos, si no se encuentran en el mismo orden, entonces su valor es F.

La función EQUIVALENCIA se aplica a dos argumentos cualesquiera y determina si ellos son iguales o equivalentes. Es decir, si se trata de dos átomos iguales, la función es T; y si se trata de dos listas que tengan los mismos elementos y únicamente ellos, entonces la función EQUIVALENCIA es T, aunque dichos elementos no se encuentren en el mismo orden.

Pues bien, las definiciones correspondientes a las funciones antes mencionadas son:

```
(DEFINE  (INTERSECCION (LAMBDA (P Q)
              (IF (NULL P) P
                 (IF (ELEMENTO (CAR P) Q)
                    (CONS (CAR P) (INTERSECCION (CDR P) Q))
                       (INTERSECCION (CDR P) Q)  )  )  ))
```

aquí, la DIFERENCIA forma la lista constituida por los elementos de P que no pertenecen a Q., y por los de Q que no están en P. Por consiguiente, en tales condiciones, la DIFERENCIA establece la lista correspondiente a la disyunción excluyente entre P y Q y, así, resulta ser una operación conmutativa.

```
(UNION (LAMBDA (P Q)
   (IF (NULL P) Q
      (IF (ELEMENTO (CAR P) Q)
(CONS (CAR P) (UNION (ELIMINA (CAR P) (CDR P))
                           (ELIMINA (CAR P) Q) ) )
         (CONS (CAR P) (UNION (CDR P) Q)) ) ) ))

(DIFERENCIA (LAMBDA (P Q)
   (IF (NULL P) Q
      (IF (ELEMENTO (CAR P) Q)
         (DIFERENCIA (CDR P) (ELIMINA (CAR P) Q) )
        (CONS (CAR P) (DIFERENCIA (CDR P) Q)) )) ))

(COMPLEMENTO (LAMBDA (P)
   (DIFERENCIA P (UNIVERSOXYZ) ) ))

(ELEMENTO (LAMBDA (P Q)
   (IF (NULL Q) (OR)
      (IF (IGUAL P (CAR Q)) (AND)
         (ELEMENTO P (CDR Q)) ) ) ))

(ELIMINA (LABDA (P Q)
   (IF (NULL Q) Q
      (IF (IGUAL P (CAR Q))
         (ELIMINA P (CDR Q))
          (CONS (CAR Q) (ELIMINA P (CDR Q))) ) ) ))

(IGUAL (LAMBDA (P Q)
   (IF (ATOM P) (EQ P Q)
      (IF (ATOM Q) (OR)
         (IF (NULL P) (NULL Q)
            (IF (NULL Q) (OR)
               (IF (IGUAL (CAR P) (CAR Q))
                  (IGUAL (CDR P) (CDR Q))
                     (OR) ) ) ) ) ) ))

(EQUIVALENCIA (LAMBDA (P Q)
   (IF (ATOM P) (EQ P Q)
      (IF (ATOM Q) (OR)
         (IF (NULL P) (NULL Q)
            (IF (NULL Q) (OR)
               (IF (ELEMENTO (CAR P) Q)
                  (EQUIVALENCIA (CDR P)
                     (ELIMINA (CAR P) Q) )
                        (OR) ) ) ) ) ) ))
```

9. *Propiedades de las listas proposicionales*

Las listas proposicionales tienen propiedades análogas a las funciones lógicas y los conjuntos, como lo mostramos a continuación con las más conocidas. Con la expresión en que aparecen, constituyen una aplicación de las definiciones que ya hemos establecido. De esta manera, su formulación es la siguiente:

```
(APPLY EQUIVALENCIA (INTERSECCION P P)      P)
(APPLY EQUIVALENCIA (UNION P P)      P)
(APPLY EQUIVALENCIA (DIFERENCIA P P) (ENANTIOSISXYZ)  )

(APPLY EQUIVALENCIA (INTERSECCION P
                    (INTERSECCION Q R)  )
                    (INTERSECCION
                    (INTERSECCION P Q)R)  )
(APPLY EQUIVALENCIA (UNION P (UNION Q R)  )
                    (UNION (UNION P Q) R)  )
(APPLY EQUIVALENCIA (DIFERENCIA P (DIFERENCIA Q R)  )
                    (DIFERENCIA (DIFERENCIA P Q) R)  )

(APPLY EQUIVALENCIA (INTERSECCION P Q)
                    (INTERSECCION Q P)  )
(APPLY EQUIVALENCIA (UNION P Q) (UNION Q P)  )
(APPLY EQUIVALENCIA (DIFERENCIA P Q) (DIFERENCIA Q P) )

(APPLY EQUIVALENCIA (INTERSECCION P (UNION Q R)  )
          (UNION (INTERSECCION P Q) (INTERSECCION P R))  )
(APPLY EQUIVALENCIA (INTERSECCION P (DIFERENCIA Q R)  )
      (DIFERENCIA (INTERSECCION P Q) (INTERSECCION P R))  )
(APPLY EQUIVALENCIA (UNION P (INTERSECCION Q R)  )
                 (INTERSECCION (UNION P Q) (UNION P R))  )
(APPLY EQUIVALENCIA (UNION P (DIFERENCIA Q R)  )
                  (DIFERENCIA (UNION P Q) (UNION P R))  )
(APPLY EQUIVALENCIA (DIFERENCIA P (INTERSECCION Q R)  )
        (INTERSECCION (DIFERENCIA P Q) (DIFERENCIA P R))  )
(APPLY EQUIVALENCIA (DIFERENCIA P (UNION Q R)  )
              (UNION (DIFERENCIA P Q) (DIFERENCIA P R))  )

(APPLY EQUIVALENCIA (INTERSECCION P (UNIVERSOXYZ))  P)
(APPLY EQUIVALENCIA (INTERSECCION P (ENANTIOSISXYZ)  )
                    (ENANTIOSISXYZ)  )
(APPLY EQUIVALENCIA (UNION P (UNIVERSOXYZ)  )
                    (UNIVERSOXYZ)  )
```

```
(APPLY EQUIVALENCIA (UNION P (ENANTIOSISXYZ)) P)
(APPLY EQUIVALENCIA (DIFERENCIA P (UNIVERSOXYZ) )
                    (COMPLEMENTO P) )
(APPLY EQUIVALENCIA (DIFERENCIA P (ENANTIOSISXYZ)) P)
(APPLY EQUIVALENCIA (INTERSECCION (UNIVERSOXYZ)
                    (ENANTIOSISXYZ))
                    (ENANTIOSISXYZ) )
(APPLY EQUIVALENCIA (UNION (UNIVERSOXYZ)
                    (ENANTIOSISXYZ)) (UNIVERSOXYZ))
(APPLY EQUIVALENCIA (DIFERENCIA (UNIVERSOXYZ)
                    (ENANTIOSISXYZ)) (UNIVERSOXYZ))
(APPLY EQUIVALENCIA (INTERSECCION P (COMPLEMENTO P) )
                    (ENANTIOSISXYZ) )
(APPLY EQUIVALENCIA (UNION P (COMPLEMENTO P) )
                    (UNIVERSOXYZ) )
(APPLY EQUIVALENCIA (DIFERENCIA P (COMPLEMENTO P) )
                    (UNIVERSOXYZ) )
(APPLY EQUIVALENCIA (COMPLEMENTO
                    (COMPLEMENTO P) ) P)
(APPLY EQUIVALENCIA (COMPLEMENTO (UNIVERSOXYZ) )
                    (ENANTIOSISXYZ) )
(APPLY EQUIVALENCIA (COMPLEMENTO (ENANTIOSISXYZ) )
                    (UNIVERSOXYZ) )
(APPLY EQUIVALENCIA (COMPLEMENTO
                    (INTERSECCION P Q) )
          (UNION (COMPLEMENTO P) (COMPLEMENTO Q)) )
(APPLY EQUIVALENCIA (COMPLEMENTO (UNION P Q) )
     (INTERSECCION (COMPLEMENTO P) (COMPLEMENTO Q)) )
(APPLY EQUIVALENCIA (DIFERENCIA P Q)
      (DIFERENCIA (COMPLEMENTO P) (COMPLEMENTO Q)) )
(APPLY EQUIVALENCIA (INTERSECCION (INTERSECCION P Q)
                                  (DIFERENCIA P Q) )
                                  (ENANTIOSISXYZ) )
(APPLY EQUIVALENCIA (UNION (INTERSECCION P Q)
                                  (DIFERENCIA P Q) )
                                        (UNION P Q) )
(APPLY EQUIVALENCIA (DIFERENCIA (INTERSECCION P Q)
                                (UNION P Q) )
                                  (DIFERENCIA P Q) )
(APPLY EQUIVALENCIA (INTERSECCION (INTERSECCION P Q)
                                  (UNION P Q) )
                                (INTERSECCION P Q) )
```

```
(APPLY  EQUIVALENCIA  (UNION (INTERSECCION P Q)
                                        (UNION P Q)  )
                                        (UNION P Q)  )
(APPLY  EQUIVALENCIA  (DIFERENCIA (INTERSECCION P Q)
                                  (DIFERENCIA  P Q)  )
                                        (UNION P Q)  )
(APPLY  EQUIVALENCIA  (INTERSECCION (UNION P Q)
                                    (DIFERENCIA P Q)  )
                                     (DIFERENCIA P Q)  )
(APPLY  EQUIVALENCIA  (UNION (UNION P Q)
                          (DIFERENCIA P Q)  )
                                        (UNION P Q)  )
(APPLY  EQUIVALENCIA  (DIFERENCIA (UNION P Q)
                                  (DIFERENCIA P Q)  )
                                   (INTERSECCION P Q)  )

FINENT)
```

En todos los caso anteriores se obtiene el resultado T en la computadora.

10. *Cálculo proposicional*

La aplicación de la función INTERSECCION a una lista proposicional en X y Y, y una lista proposicional en Y y Z, produce como resultado otra lista proposicional que, interpretada respecto a X y Z, representa la conclusión de la inferencia mediata que tiene como premisas las dos primeras listas proposicionales. A continuación examinamos los casos típicos entre los 100 que son posibles.

Del primer tipo de inferencia mediata, cuando las dos premisas son proposiciones universales definidas, tenemos como ejemplo de sus 4 casos, el siguiente:

(APPLY INTERSECCION (RECIPROCIDADXY) (RECIPROCIDADYZ))
de donde se obtiene la lista:

((X Y Z) (NX NY NZ))

de la que, eliminando los parámetros Y y NY, nos queda:

((X Z) (NX NZ))

que representa la lista proposicional correspondiente a la RECIPROCIDADZX.

De los 16 casos del segundo tipo de inferencia mediata, cuando una premisa es una proposición universal definida y la otra es una proposición universal indefinida, tenemos como ejemplo:

(APPLY INTERSECCION (RECIPROCIDADXY) (IMPLICACIONYZ))

de donde se obtiene la lista:

((X Y Z) (NX NY Z) (NX NY NZ))

de la que, eliminando los parámetros Y y NY, resulta:

((X Z) (NX Z) (NX NZ))

que representa la IMPLICACIONXZ.

Del tercer tipo de inferencia mediata, con una premisa universal definida y la otra premisa particular, que comprende 16 casos, tomamos como ejemplo:

(APPLY INTERSECCION (EXCLUSIONXY) (DISCORDANCIAYZ))

de la que resulta la lista:

((NX Y NZ))

de la cual se obtiene, eliminando el parámetro Y:

((NX NZ))

que expresa la HETEROFASISXZ.

Del cuarto tipo de inferencia mediata, con sus 8 casos, en donde las dos premisas son proposiciones universales indefinidas, tenemos como ejemplo:

(APPLY INTERSECCION
(INCLUSIONXY) (INCOMPATIBILIDADYZ))

de la que se obtiene la lista:

((X Y NZ) (X NY Z) (X NY NZ) (NX Y NZ))

de donde resulta, eliminando los parámetros Y y NY. lo mismo que el elemento (X NZ) que se encuentra repetido:

((X NZ) (X Z) (NX NZ))

que es equivalente a la IMPLICACIONZX.

Del quinto tipo de inferencia mediata, en el cual las dos premisas son también proposiciones universales indefinidas, tenemos como ejemplo de sus 8 casos:

(APPLY INTERSECCION (IMPLICACIONYX) (IMPLICACIONYZ))

de donde resulta la lista:

((X Y Z) (X NY Z) (X NY NZ) (NX NY Z) (NX NY NZ))

de donde se obtiene, eliminando los parámetros Y y NY:

((X Z) (X Z) (X NZ) (NX Z) (NX NZ))

la cual, por incluir los cuatro elementos correspondientes al UNIVERSOZXY, es equivalente a la lista:

((X Z))

que representa la CONJUNCIONXZ.

Del sexto tipo de inferencia mediata, que tiene como premisa una proposición universal indefinida y una particular, incluyendo 16 casos, tenemos como ejemplo:

(APPLY INTERSECCION (IMPLICACIONYX) CONJUNCIONYZ))

se obtiene la lista:

((X Y Z))

de la que, eliminando el parámetro Y, queda:

((X Z))

que representa la CONJUNCIONXZ.

Además, resulta también un séptimo tipo de inferencia mediata, que tiene como premisas dos proposiciones particulares,[16] y comprende los 8 casos siguientes:

(APPLY INTERSECCION (CONJUNCIONXY) (CONJUNCIONYZ))
(APPLY INTERSECCION (CONJUNCIONXY) (DISCORDANCIAYZ))
(APPLY INTERSECCION (DISCORDANCIAXY)
(DISCORDANCIAZY))
(APPLY INTERSECCION (DISCORDANCIAXY)
(HETEROFASISYZ))
(APPLY INTERSECCION (DISCORDANCIAYX) (CONJUNCIONYZ))
(APPLY INTERSECCION (DISCORDANCIAYX)
(DISCORDANCIAYZ))
(APPLY INTERSECCION (HETEROFASISXY)
(DISCORDANCIAZY))
(APPLY INTERSECCION (HETEROFASISXY)
(HETEROFASISYZ))

[16] Estas 8 inferencias fueron descubiertas por Pastore en su máquina lógica (cf. *op. cit.*). En efecto, de las premisas "Algunos X son algunos Y" y "Algunos Y son algunos Z", se concluye válidamente que "Algunos X son algunos Z", con tal que los "algunos" de la conclusión sean los mismos "algunos" de ambas premisas.

Efectivamente, tomando como ejemplo el primer caso, se obtiene la lista:

((X Y Z))

de donde, eliminando el parámetro Y, se tiene:

((X Z))

que representa la CONJUNCIONXZ.

Por otra parte, existen 16 casos en que la INTERSECCION de una proposición universal indefinida y una particular no es conclusiva, debido a que la proposición particular es un subconjunto de la proposición universal y, por ende, el resultado que se obtiene es la misma proposición particular. Como ejemplo tenemos el siguiente:

(APPLY INTERSECCION (IMPLICACIONXY)
(DISCORDANCIAYZ))

de donde resulta la lista:

((X Y NZ) (NX Y NZ))

que es precisamente la lista correspondiente a la DISCORDANCIAYZ.

En fin, hay 8 casos en que la aplicación de la INTERSECCION a dos proposiciones particulares produce como resultado una lista nula, debido a que las listas proposicionales respectivas son disyuntas. Así tenemos, por ejemplo:

(APPLY INTERSECCION (DISCORDANCIAXY) (CONJUNCIONYZ))

que da como resultado la lista:

( )

que representa la ENANTIOSISZXY.

De esta manera, hemos examinado los 100 casos posibles de la inferencia mediata.

Por lo que se refiere a las inferencias disyuntivas y condicionales, su conclusión se obtiene por medio del COMPLEMENTO de la DIFERENCIA. En efecto, como ejemplo de los 8 casos de inferencia por afirmación, tenemos el siguiente:

(APPLY COMPLEMENTO (DIFERENCIA (RECIPROCIDADXY)
(PROFASISXY)))

de donde se obtiene la lista proposicional:

((X Y Z) (X Y NZ) (NX Y Z) (NX Y NZ))

que es la expresión de la PROFASISYX.

Asimismo, de los 8 casos de inferencia por negación, tenemos el siguiente ejemplo:

(APPLY COMPLEMENTO (DIFERENCIA (EXCLUSIONXY)
(ANTIFASISYX)))

de donde resulta la lista:

((X Y Z) (X Y NZ) (X NY Z) (X NY NZ))

que representa la PROFASISXY.

En cuanto a los dilemas, sus 12 casos posibles resultan de la aplicación de la función INTERSECCION a tres proposiciones universales, como se ilustra en el ejemplo que sigue:

(APPLY INTERSECCION (INCLUSIONXY)
(INTERSECCION (IMPLICACIONXZ) (IMPLICACIONYZ)))

de la cual resulta la lista:

((X Y Z) (X NY Z) (NX Y Z))

de donde, eliminando los parámetros Y y NY, lo mismo que el elemento (X Z) que se repite, tenemos:

((X Z) (NX Z))

que representa la PROFASISZX.

Finalmente nos referimos a las inferencias inductivas por amplificación.

La INTERSECCION de dos proposiciones singulares que no sean contradictorias, o sea, recíprocamente complementarias, produce una proposición particular, como por ejemplo:

(APPLY INTERSECCION (PROFASISXY) (PROFASISYX))

da como resultado la lista:

((X Y Z) (X Y NZ))

que corresponde a la CONJUNCIONXY.

La UNION de dos proposiciones singulares que no sean recíprocamente complementarias, produce una proposición universal indefinida, como por ejemplo:

(APPLY UNION (PROFASISYX) (ANTIFASISXY))

de la que se obtiene como resultado la lista:

((X Y Z) (X Y NZ) (NX Y Z) (NX Y NZ) (NX NY Z) (NX NY NZ))

que corresponde a la IMPLICACIONXY.

La UNION de tres proposiciones particulares produce siempre una proposición universal indefinida, como se muestra en el ejemplo siguiente:

(APPLY UNION (CONJUNCIONXY)
(UNION (DISCORDANCIAXY) (DISCORDANCIAYX)))

que da como resultado la lista:

((X Y Z) (X Y NZ) (X NY Z) (X NY NZ) (NX Y Z) (NX Y NZ))

que corresponde a la INCLUSIONXY.

La UNION de dos proposiciones particulares que no sean subcontrarias, produce una proposición universal definida, como se ilustra con el siguiente ejemplo:

(APPLY UNION (CONJUNCIONXY) (HETEROFASISXY))

de donde se obtiene la lista:

((X Y Z) (X Y NZ) (NX NY Z) (NX NY NZ))

que corresponde a la RECIPROCIDADXY.

La INTERSECCION de dos proposiciones universales indefinidas que no sean contrarias, produce como resultado una proposición universal definida, como se muestra en el ejemplo siguiente:

(APPLY INTERSECCION (INCLUSIONXY)
(INCOMPATIBILIDADXY))

que da como resultado la lista:

((X NY Z) (X NY NZ) (NX Y Z) (NX Y NZ))

que corresponde a la EXCLUSIONXY.

La UNION de dos proposiciones recíprocamente complementarias produce como resultado la lista universal, como se ilustra en el ejemplo que sigue:

```
(APPLY  UNION  (PROFASISXY)  (ANTIFASISXY)  )
```

que da como resultado la lista:

```
(  (X Y Z)  (X Y NZ)  (X NY Z)  (X NY NZ)  (NX Y Z)  (NX Y NZ)
                (NX NY Z)  (NX NY NZ)  )
```

que corresponde al UNIVERSOXYZ.

La UNION de las cuatro proposiciones particulares produce como resultado la lista universal, como se muestra en seguida:

```
(APPLY  UNION  (CONJUNCIONXY)
                        (UNION  (DISCORDANCIAXY)
                            (UNION  (DISCORDANCIAYX)
                                    (HETEROFASISXY))  )  )
```

que da como resultado la lista:

```
(  (X Y Z)  (X Y NZ)  (X NY Z)  (X NY NZ)  (NX Y Z)  (NX Y NZ)
                (NX NY Z)  (NX NY NZ)  )
```

que corresponde al UNIVERSOXYZ.

Por último, la UNION de dos proposiciones universales, definidas o indefinidas, produce como resultado la lista universal, como se ilustra en el ejemplo que sigue:

```
(APPLY  UNION  (IMPLICACIONXY)  (IMPLICACIONYX)  )
```

que da como resultado la lista:

```
(  (X Y Z)  (X Y NZ)  (X NY Z)  (X NY NZ)  (NX Y Z)  (NX Y NZ)
                (NX NY Z)  (NX NY NZ)  )
```

que corresponde al UNIVERSOXYZ.

Con lo anterior esperamos haber mostrado algunas de las posibilidades que ofrece el tratamiento de las proposiciones expresadas en forma de listas y la aplicación que a ellas puede hacerse de las operaciones correspondientes a los conjuntos.

ELI DE GORTARI