NOTAS Y COMENTARIOS

CONFECCION AUTOMATICA DE TABLAS DE VERDAD

Rafael Beneyto
Universidad de Valencia

1. Operadores lógicos y lenguaje BASIC

La casi totalidad de los lenguajes de programación hoy más disponibles se hallan típicamente orientados hacia el cálculo matemático. Ello, sin embargo, no impide aplicarlos provechosamente a la resolución de problemas lógicos. Incluso en procesos de manipulación simbólica.[1] Pero las tablas de verdad constituyen, en cierta medida, un sistema cercano a los cálculos matemáticos. La expresión "$P \wedge Q$" ("P y Q"), por ejemplo, toma como valor el resultado de llevar a cabo la operación indicada por "$\wedge$" sobre los valores de las expresiones "P" y "Q". Es éste el espíritu de los operadores "AND", "OR" y "NOT" que encontramos en lenguajes como BASIC. Y dado que "si... entonces..." y "... si y sólo si..." son definibles en términos de los anteriores, se puede acudir al BASIC para la confección mecánica por computador de las tablas de verdad del cálculo de enunciados.

El BASIC[2] tiene también disponibles recursos que nos dispensan, sin embargo, de la tarea de traducir los enunciados lógicos a otros en los que los juntores lógicos se reduzcan a "AND", "OR"

[1] Un ejemplo de ello es el programa que computa el análisis veritativo-funcional. *Cf.* R. Beneyto: "El análisis veritativo-funcional de Quine: computación de un algoritmo lógico", en *Aspectos de la filosofía de Quine* (*Teorema*, número monográfico, 1975).

[2] Para una mayor y más analizada noticia de la aplicación del BASIC a la computación de la lógica, véase el *Anexo* introducido en la nueva edición del libro de M. Garrido: *Lógica simbólica* (Madrid: Tecnos, 1977). En él también puede el lector encontrar un extracto del BASIC y una presentación en lenguaje LISP del programa con que Hao Wang computó los teoremas de la lógica proposicional contenidos en *Principia Mathematica*.

y "NOT". Considérese el operador relacional "$\leqslant$" (menor o igual que) cuando los argumentos a comparar toman exclusivamente los valores "1" y "0" –como es el caso de los enunciados lógicos bivalentes. En la siguiente tabla se podrá observar que la relación "$\leqslant$" entre P y Q se satisface –toma el valor "1"– exactamente en los mismos casos en que es verdadero –toma el valor "1"– el enunciado "$P \rightarrow Q$":

P	Q	$P \leqslant Q$	$P \rightarrow Q$	$P = Q$	$P \leftrightarrow Q$	$P * Q$	$P \wedge Q$
1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	0	1	1	1	1	0	0

En situación similar se encuentra el coimplicador "$\leftrightarrow$" respecto del operador de igualdad "$=$" y el conjuntor "$\wedge$" con relación al operador del producto "$*$". Razón por la que sustituiremos estos últimos juntores por los operadores BASIC a ellos correlatados. De donde resulta que los juntores lógicos que utilizaremos en BASIC son

NOT: el negador
$\leqslant$: el implicador
$=$: el coimplicador
$*$: el conjuntor
OR: el disyuntor

ordenados por prioridad de ejecución –menor fuerza lógica.[3] Como es de sospechar, el orden natural de ejecución puede alterarse con la ayuda de paréntesis. Así

NOT $P * Q \leqslant R = S$ es el enunciado
$\neg P \wedge ((Q \rightarrow R) \leftrightarrow S)$ mientras que
NOT $((P * Q) \leqslant (R = S))$ es el enunciado
$\neg ((P \wedge Q) \rightarrow (R \leftrightarrow S))$.

¿Cuál es el valor de la expresión escrita en primer lugar? Al

[3] Con la salvedad de que "$\leqslant$" y "$=$" tienen la misma fuerza y se evalúan prioritariamente según su orden de aparición de izquierda a derecha.

igual que nos ocurre a nosotros en la confección de las tablas, el computador sólo puede evaluarla cuando se le han dado valores a cada una de sus variables, pudiendo la evaluación de la expresión variar con los valores de éstas. Una fase previa, pues, a la misma ha de ocuparse de asignar adecuadamente valores "1" y "0" a las distintas variables. El siguiente sub-programa BASIC [4] conduce al resultado que se indica a su derecha.

```
FOR P = 0 TO 1
  FOR Q = 0 TO 1
  NEXT Q
NEXT P
```

P	Q
0	0
0	1
1	0
1	1

Resultado en el que se contemplan, sin repetición, todas las posibles combinaciones de dichos valores. Una técnica, pues, consistiría en introducir en secuencia una instrucción "FOR" para cada una de las distintas letras enunciativas *P, Q, R,* ... y una secuencia, en orden inverso, de enunciados "NEXT" para las mismas. Si entre ambas secuencias se ordenara la impresión del valor del enunciado problema, obtendríamos como resultado la impresión de dicho valor para cada una de las líneas de la correspondiente tabla.

2. Un programa autónomo

Este es el espíritu de los programas de Kemeny-Kurtz[5] y de Purtill[6] –excepto en lo que atañe a la utilización de "*", "≤" y "=" como operadores lógicos. La simplicidad de los mismos acarrea, por contra, un inconveniente. Tal es que la lista de instrucciones "FOR –NEXT" está en función de las letras enunciativas que concretamente constituyen el enunciado cuya tabla deseamos construir. Mi intención ahora es presentar un programa que salva dicho inconveniente, aunque para ello debamos renunciar a una técnica

[4] En este programa, para cada uno de los dos valores "0" y "1" de *P, Q* toma alternativamente los dos valores "0" y "1"

[5] J.G. Kemeny & Th.E. Kurtz: BASIC *Programming* (New York: John Wiley & Sons, Inc., 1967, 2ª ed.), pp. 30–31.

[6] R.L. Purtill: "Doing Logic by Computer", en *Notre Dame Journal of Formal Logic*, vol. X, nº 2, Abril, 1969, pp. 150–162.

tan simple y elegante Esta que yo ahora presento requiere utilizar "P(1)", "P(2)", ... en lugar de las letras enunciativas "*P*", "*Q*", ... ("*P*" será en el programa un vector con 100 elementos; cada uno de ellos puede sustituir a una letra enunciativa, por lo que es preciso especificar en cada caso cuántas letras enunciativas diferentes contiene el problema –lo cual se indica en la instrucción "INPUT N1", donde "N1" es el número de tales letras). Ello nos permitirá elaborar un programa autónomo en el siguiente sentido. Nuestro programa es aplicable a la solución de no importa qué problema del cálculo de enunciados. Para ello basta en cada caso especificar qué enunciado se quiere computar e indicar cuántas letras enunciativas diferentes contiene.

Mas en tales circunstancias, no nos podemos servir de los nidos "FOR–NEXT" para la combinación de valores de los átomos lógicos. Ahora bien, aunque nos está vedado recurrir a esta posibilidad técnica, sí podemos *simularla*. Partamos de una situación inicial en la que el valor de los diversos átomos es cero: "MAT P = ZER". En este momento estamos en condiciones de evaluar la primera línea de la tabla, lo cual se hace en la rutina de impresión. Sea *N1* el número de variables del enunciado. Sea *I1* un índice que toma valores desde 1 hasta *N1*, y cuya función consiste en facilitarnos la referencia a los diversos átomos *P(I1)*. Podemos hacer que en el momento de la impresión *I1* sea igual a *N1*. Considérese ahora la siguiente lista de instrucciones:

1. Dar el valor 0 a todo el vector.
2. Dar a *I1* el valor *N1*.
3. Pasar a 13.
4. Aumentar en 1 el valor de *P(I1)*.
5. Si *P(I1)* vale 1 pasar a la línea 9.
6. Si *I1* vale 1 pasar a la línea 15.
7. Disminuir en 1 el valor de *I1*.
8. Pasar a la línea 4.
9. Si *I1* vale *N1* pasar a la línea 13.
10. Aumentar en 1 el valor de *I1*.
11. Hacer *P(I1)* igual a 0.
12. Pasar a la línea 9.
13. Evaluar el enunciado y escribir el resultado.
14. Pasar a la línea 4.
15. Fin.

La serie de instrucciones 4–14 parte de la última evaluación practicada para los átomos en la tabla. Retrocede átomo a átomo hasta encontrar uno al que no se le haya asignado en la línea el valor 1. Se le asigna a éste el valor 1, se mantiene el valor de los átomos anteriores y se asigna 0 a los posteriores, consiguiéndose así la combinación de valores correspondiente a la nueva línea de la tabla. Sólo en el momento en que en el proceso se ha alcanzado el primer átomo sin que se haya encontrado uno de ellos que tenga el valor 0 en la combinación de valores precedente, se da por concluida la confección de la tabla.

Sólo falta para obtener el resultado apetecido introducir en el momento oportuno la orden de impresión del valor de la tabla. Ello debe hacerse cuando se ha concluido la evaluación para una de las líneas, lo cual acontece cuando *I1* alcanza el valor *N1*.

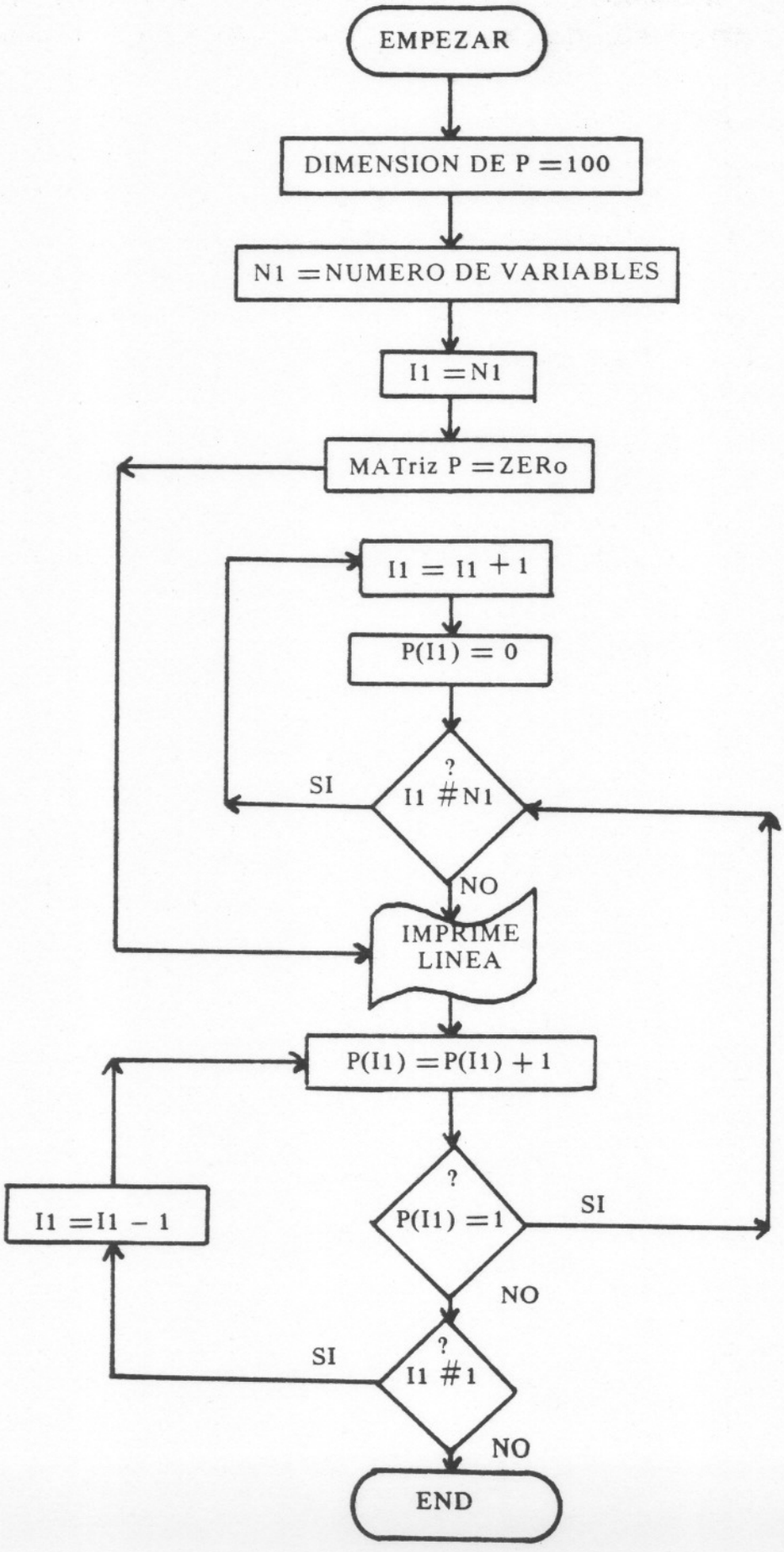
A continuación presentamos un diagrama de flujo para la confección automática de tablas de verdad y un programa, construido en BASIC, para el mismo.

Si establecemos que las letras enunciativas del alfabeto lógico constituyen el conjunto ordenado

$$\langle P, Q, R, \ldots, Y, Z, A, B, \ldots, M, N, O \rangle$$

y que todo enunciado de n letras enunciativas distintas cuya tabla se desea confeccionar está construido con las n primeras letras de dicho conjunto, podemos eludir el condicionamiento antes mencionado de que los enunciados se presenten al computador en términos de "$P(1)$", "$P(2)$", etc. Para ello basta adicionar al programa las siguientes 26 instrucciones

```
1001 LET P = P(1)
  .
  .
1011 LET Z = P(10)
1012 LET A = P(11)
  .
  .
1026 LET O = P(26)
```


EMPEZAR
DIMENSION DE P =100
N1 =NUMERO DE VARIABLES
I1 =N1
MATriz P =ZERo
I1 = I1 + 1
P(I1) = 0
?
I1 #N1
SI
NO
IMPRIME
LINEA
P(I1) =P(I1) + 1
?
P(I1) =1
SI
NO
I1 =I1 - 1
?
I1 #1
SI
NO
END

```
1  REM *** TABLAS DE VERDAD EN BASIC
2  REM *** '*' ES EL CONJUNTOR, '<=' EL IMPLICADOR,
3  REM ***'=' EL COIMPLICADOR, 'NOT' ES EL NEGADOR Y
4  REM ***'OR' ES EL DISYUNTOR. LOS DISTINTOS ATOMOS
5  REM ***SON P(1), P(2),... O BIEN P, Q, ...
6  REM ***EL ENUNCIADO CUYA TABLA SE DESEA CON-
7  REM *** FECCIONAR SE INTRODUCE ORDENANDO AL
8  REM *** COMPUTADOR QUE IMPRIMA SU VALOR EN LA
9  REM *** LINEA 2001 Y SUCESIVAS HASTA LA 3000.
10 REM *** CUANDO EL ENUNCIADO ES EXCESIVAMENTE
11 REM *** LARGO Y NO PUEDE AJUSTARSE A LAS DI-
12 REM *** MENSIONES DE UNA LINEA DEL COMPUTADOR
13 REM *** DEBE DESCOMPONERSE EN OTROS ENUNCIA-
14 REM *** DOS ANTES DE ORDENAR QUE SE IMPRIMA
15 REM *** SU VALOR, APROVECHANDO PARA ELLO LAS
16 REM *** LINEAS ANTES MENCIONADAS
17 REM *** POR EJEMPLO EL ENUNCIADO
18 REM  ((P(1) <= P(2)) * ((P(2) = NOT(P(1)))
19 REM *** SE DESCOMPONE EN LOS ENUNCIADOS
20 REM ***  2001 LET P9 = P(1) <= P(2)
21 REM ***  2002 LET P8 = (P(2) = NOT P(1))
22 REM *** Y ORDENAR LO SIGUIENTE
23 REM ***   2003 PRINT P9 * P8
24 REM *** PUESTO QUE AL VECTOR P LE HEMOS ASIG-
25 REM *** NADO UNA DIMENSION DE SOLO 100, PODE-
26 REM *** MOS CONFECCIONAR TABLAS DE ENUNCIA-
27 REM *** DOS CON 100 LETRAS ENUNCIATIVAS DIFE-
28 REM *** RENTES (A LO SUMO). 'I1' ES UNA VARIABLE
29 REM *** QUE DETERMINA A QUE LETRA ENUNCIATIVA
30 REM *** SE LE VA A ASIGNAR VALORES
31 DIM P(100)
32 PRINT "CUANTAS VARIABLES DIFERENTES HAY";
33 INPUT N1
34 LET I1 = N1
35 MAT P = ZER
36 GOTO 90
70 LET I1 = I1 + 1
73 LET P(I1) = 0
75 IF (I1 # N1) THEN 70
```

```
90  FOR J1 = 1 TO N1
92  PRINT P(J1);
95  NEXT J1
1900  PRINT "PROBL. = ";
1950  REM **A PARTIR DE LA LINEA 2000 SE ORDENA QUE
1951  REM ** IMPRIMA EL VALOR DE LA FORMULA CUYA
1952  REM ** TABLA SE DESEA COMPUTAR
3001  LET P(I1) = P(I1) +1
3002  IF P(I1) = 1 THEN 75
3003  IF I1 = 1 THEN 4000
3004  LET I1 = I1 - 1
3005  GOTO 3001
4000  END

READY
```