

Apuntes de Lógica Digital

Daniel Araya Román

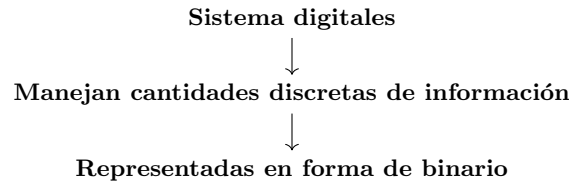
2023-05-08

1. Sistemas Binarios

1.1 Sistemas Digitales

En los sistemas digitales electrónicos actuales, las señales emplean sólo dos valores discretos \rightarrow *binarios*. Un dígito binario, llamado **bit**, que puede tomar los valores 0 y 1. Un sistema digital es una interconexión de módulos digitales. Para entender como funciona cada módulo digital, se necesitan conocimientos básicos de circuitos digitales y de su función lógica.

Un lenguaje importante para el diseño digital es el (**HDL, Hardware Description Language**). Sirve para simular sistemas digitales y verificar su funcionamiento antes de crearlos en hardware.



1.2 Números Binarios

El número decimal 7392, contiene potencias de 10 que están implícitas en la posición de los coeficientes, e.g:

$$7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

Un número con punto decimal se representa con una serie de coeficientes, así:

$$a_5 a_4 a_3 a_2 a_1 a_0 \cdot a_{-1} a_{-2} a_{-3}$$

los coeficientes a_j son cualesquiera de los 10 dígitos (0...9); el valor del subíndice j indica la posición, y la potencia de 10 que se deberá multiplicar ese coeficiente. De modo que:

$$10^5 a_5 + 10^4 a_4 + 10^3 a_3 + 10^2 a_2 + 10^1 a_1 + 10^0 a_0 + 10^{-1} a_{-1} + 10^{-2} a_{-2} + 10^{-3} a_{-3}$$

El sistema binario es distinto al decimal, sus coeficientes solo pueden tener 2 valores, 0 o 1. Cada coeficiente a_j se multiplica por 2^j . 11010.11 es 26.75 en decimal, porque:

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 26.75$$

En general, un número expresado en un sistema de base r consiste en coeficientes que se multiplican por potencias de r :

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \dots + a_2 \cdot r^2 + a_1 \cdot r + a_0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \dots + a_{-m} \cdot r^{-m}$$

1.4 Números Octales y Hexadecimales

Las conversiones entre binario, octal y hexadecimal son importantes en las computadoras digitales. Puesto que $2^3 = 8$ y $2^4 = 16$, cada dígito octal corresponde a **tres** dígitos binarios, y cada dígito hexadecimal corresponde a **cuatro** dígitos binarios.

Binario \rightarrow octal: agrupando los dígitos binarios de 3 en 3, de derecha a izquierda, y reemplazando cada grupo por su equivalente octal.

$$(10\ 110\ 001\ 101\ 011 \cdot 111\ 100\ 000\ 110)_2 = (26153.7406)_8$$

Binario \rightarrow hexadecimal: agrupando los dígitos binarios de 4 en 4, de derecha a izquierda, y reemplazando cada grupo por su equivalente hexadecimal.

$$(10\ 1100\ 0110\ 1011 \cdot 1111\ 0010)_2 = (2C6B.F2)_{16}$$

Cuando se habla de binario es más deseable expresarlo en términos de números octales o hexadecimales, porque son más compactos y fáciles de leer. Así $(111\ 111\ 111\ 111)_2$ este número en binario de 12 bits, se puede escribir como $(7777)_8$ en octal o $(FFF)_{16}$ en hexadecimal.

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Table 1: Números en diferentes bases.

1.5 Complementos

En las computadoras digitales se usan complementos para simplificar la operación de resta y para efectuar manipulaciones lógicas. Existen dos tipos de complementos para cada sistema de base r : el *complemento a la base* y el *complemento a la base disminuida*. El primero se denomina complemento a r , mientras que el segundo es el complemento a $(r - 1)$. Si se sustituye el valor de la base r en los nombres obtenemos que los dos tipos son el complemento a 2 y el complemento a 1.

1.5.1 Complemento a la base

El complemento a r de un número N de n dígitos en base r se define como:

$$r^n - N, \text{ para } N \neq 0, \text{ y } 0 \text{ para } N = 0.$$

Por ejemplo, el complemento a 10 de 1234 es $10^4 - 1234 = 8766$. De forma similar, el complemento a dos se forma dejando como están todos los ceros menos significativos y el primer uno, y sustituyendo los unos por ceros y los ceros por unos en las demás posiciones a la izquierda.

El complemento a dos de 1101100 es 0010100.

El complemento a dos de 0110111 es 1001001.

1.5.2 Complemento a la base disminuida

De igual manera, dado un número N en base r que tiene n dígitos, el complemento a $(r - 1)$ de N se define como:

$$(r^n - 1) - N.$$

Con números decimales, $r = 10$ y $r - 1 = 9$, así el complemento a nueve de N es $(10^n - 1) - N$. El 10^n representado por n nueves. Por ejemplo, si $n = 4$, tenemos $10^4 = 10,000$ y $10^4 - 1 = 9999$. El complemento a nueve se consigue restando cada dígito a nueve. *e.g.*:

Complemento a nueve de 546700 es $999999 - 546700 = 453299$.

Complemento a nueve de 012398 es $999999 - 012398 = 987601$.

Ahora con números binarios, $r = 2$ y $r - 1 = 1$, así el complemento a uno de N es $(2^n - 1) - N$. En este caso 2^n se representa con un número binario que consiste en un uno seguido de n ceros.

$$n = 4, \quad 2^4 = 10000_2$$

Por otro lado $2^n - 1$ es un número binario representado por n unos.

$$n = 4, \quad 2^4 - 1 = 1111_2$$

El complemento a uno se consigue invirtiendo cada dígito. El restar dígitos binarios a uno podemos tener $1 - 1 = 0$ y $1 - 0 = 1$. Cambiando el bit de 0 a 1 o de 1 a 0. *e.g.*:

Complemento a uno de 101101 es $111111 - 101101 = 010010$.

Complemento a uno de 011010 es $111111 - 011010 = 100101$.

El complemento a $(r - 1)$ de los números octales y hexadecimales se obtiene restando cada dígito a 7 y F, respectivamente.

1.5.3 Resta con complementos

La resta de dos números de n dígitos sin signo, $M - N$, en base r se realiza así:

1. $M + (r^n - N) = M - N + r^n$
2. Si $M \geq N$, la suma produce acarreo final. Quedando $M - N$.
3. Si $M < N$, la suma no produce acarreo final. Quedando $r^n - (N - M)$.

1.6 Números binarios con signo

Por limitaciones de hardware, las computadoras deben de representar todo con dígitos binarios. Por lo tanto, los números binarios con signo se representan con un bit en la posición más significativa que se usa para indicar el signo del número. la convención es que el bit sea cero si el número es positivo y uno si es negativo.

Por ejemplo la cadena de bits 01001 se considera como 9 (binario sin signo) o +9 (binario con signo). La cadena de bits 11001 se considera como 25 (binario sin signo) o -9 (binario con signo). A esto se le llama *convención de magnitud con signo*. Así:

$$(+ \text{ o } -) \rightarrow (0 \text{ o } 1)$$

1.7 Códigos binarios

Existe una analogía directa entre:

1. Señales binarias
2. Elementos binarios
3. Dígitos binarios

Los códigos binarios solo cambian los símbolos, no el significado de los elementos.

Un código binario de n bits, es un grupo de 2^n combinaciones de 0's y 1's.

* Cada combinación representa a un elemento del conjunto codificado.

Las combinaciones de códigos de $n - bits$ se representan así:

$$C_e = [0 \quad \dots \quad 2^n - 1]$$

El número mínimo para codificar 2^n elementos es n bits. No existe un número máximo.

1.7.1 Código BCD

El código BCD \rightarrow **Binary-Coded Decimal** es un código que almacena los dígitos decimales representados de forma de dígitos binarios. Las computadoras solo entienden con valores binarios. Es posible crear distintos códigos binarios para representar 2^n combinaciones.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Table 2: Código BCD

Las combinaciones (1010 \rightarrow 1111) no se usan y por lo tanto carecen de significado, en el código BCD.

Un ejemplo de código BCD comparado con binario y decimal:

$$(185)_{10} = (10111001)_2 = (0001\ 1000\ 0101)_{BCD}$$

Es importante reiterar que $BCD \neq \text{Binario}$, BCD = Números decimales representados en forma de bits.

1.7.2 Otros códigos decimales

(BCD, 2421) \rightarrow (Códigos ponderados):

Asigna cada posición de bit \leftrightarrow Factor de ponderación (peso).

Cada dígito pueda evaluarse sumando los pesos de todos los 1's de la combinación codificada. Pesos de (BCD): 8, 4, 2, 1.

(2421, excess-3) \rightarrow (Códigos autocompletadores):

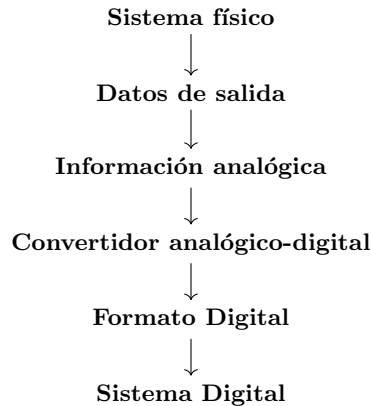
Complemento a 9 \rightarrow número decimal \rightarrow se obtiene intercambiando los 0's \rightarrow 1's y los 1's \rightarrow 0's.

1.7.3 Código Gray

Normalmente los datos de salida de **sistemas físicos** producen cantidades continuas. Por lo que se ocupa:

- Convertir las cantidades continuas en cantidades discretas.

- Convertir las cantidades discretas en cantidades digitales.
- Conviene usar el código Gray para este propósito.



La ventaja de usar el código Gray es que la diferencia entre dos números cualesquiera es únicamente de 1 bit.

(Binario) : (0111 1000), *(Gray)* : (0100 1100)

Una aplicación del código Gray es en los datos analógicos se representan mediante el cambio continuo en la posición de un eje.

1.7.4 Código ASCII

El código ASCII consta de 7-bits para su codificación, por lo que tiene un conjunto de 128 combinaciones distintas. Con $b_1 \rightarrow b_7$ siendo b_7 el bit más significativo, por ejemplo: letra A: 100 0001, (columna 100, fila 0001).

Contiene 94 caracteres imprimibles y 34 caracteres no imprimibles. Estos no imprimibles son caracteres de control.

- 26 letras minúsculas y 26 letras mayúsculas. (52)
- 10 dígitos decimales. (10)
- 32 caracteres especiales. (32)

1.7.5 Tipos de caracteres de control

1. Creadores de Formato

Controlan la forma de imprimir, controles conocidos \rightarrow máquinas de escribir.

- (BS): Retroceso
- (HT): Tabulador Horizontal

- (CR): Retorno de carro

2. Separadores de Información

Separan datos en divisiones como párrafos, líneas, páginas, etc.

- (RS): Separador de Registros
- (FS): Separador de Archivos

3. Controladores de Información

Transmisión de datos entre terminales remotas.

- (STX): Inicio de Texto
- (ETX): Fin de Texto

Encuadran un mensaje entre líneas telefónicas.

(8 bits) \rightarrow (1 byte) \rightarrow (1 carácter ASCII),
Se almacena 1 Byte por carácter ASCII.

1.7.6 Código para detectar errores

Para poder detectar errores en la comunicación de datos, se agrega un bit extra.
Este bit indica la paridad:

(Paridad): Se refiere a la cantidad de bits 1's en un byte. Si la cantidad de bits 1's es par, se le asigna un 0, si es impar, se le asigna un 1. Es más común la paridad par.

El bit de paridad se transmite,
el receptor verifica la paridad del byte recibido,
si la paridad es correcta, se asume que no hay error,
si la paridad es incorrecta, se asume que hay un error.

1.8 Almacenamiento Binario y Registros

La información binaria \rightarrow debe existir físicamente,
medio de almacenamiento en bits \rightarrow registro,
un registro de $\rightarrow n$ bits.

El estado de un registro es una tupla de n bits;
el contenido \rightarrow función, la interpretación \rightarrow información.

Un registro de 16 bits:

1100001111001001

Un registro puede:

- **Almacenar datos**
elementos discretos de información.

- **Almacenar instrucciones**

misma configuración de bits, distinta interpretación.

Manipulación de variables binarias \rightarrow circuitos lógicos digitales.

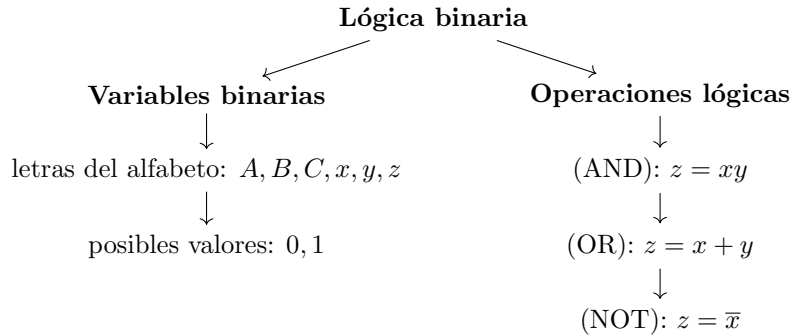
Transferencia de registros \rightarrow operación básica en sistemas digitales.

1.9 Lógica Binaria

Variables \rightarrow 2 valores discretos,

Operaciones \rightarrow 3 operaciones lógicas.

Se habla en términos de bits,
en valores de (0's y 1's), álgebra booleana.



1.9.1 Distintas interpretaciones

aritmética binaria:

$$1 + 1 = 10$$

lógica binaria:

$$1 + 1 = 1$$

1.9.2 Compuertas lógicas

Son dispositivos que operan 1 o más entradas binarias para producir una salida binaria.

(AND), (OR), (NOT)

2. Álgebra booleana y compuertas lógicas

2.1 Definiciones básicas

En el álgebra booleana, al igual que en todos los sistemas matemáticos deductivos, se define con un conjunto de elementos, un conjunto de operadores y varios axiomas. Un conjunto de elementos es cualquier colección de objetos con alguna propiedad en común.

2.1.1 Conjunto de elementos

Si S es un conjunto y x y y son ciertos objetos, entonces $x \in S$, se denota que x es un miembro del conjunto S , S , y $y \notin S$, denota que y no es un miembro del conjunto S . $A = \{1, 2, 3, 4\}$, denota que estos elementos son miembros del conjunto A .

2.1.2 Conjunto de operadores

Un operador binario definido sobre un conjunto S de elementos es una regla que asigna a cada par de elementos de S un elemento único de S . Por ejemplo, $a * b = c$. Se designa $*$ como operador binario si especifica una regla para encontrar c a partir del par (a, b) y además si $a, b, c \in S$. Por contraparte no se designa operador binario si se descubre que $a, b \in S$, pero $c \notin S$.

2.1.3 Conjunto de axiomas

1. **Cerradura.** Un conjunto S es cerrado respecto a un operador binario si, por cada par de elementos de S , el operador especifica una regla para obtener un elemento único de S . Por ejemplo, el conjunto de números naturales $N = \{1, 2, 3, \dots\}$ el operador binario más (+). Pero no es cerrado respecto al operador binario menos (-), por las reglas de la resta aritmética.
2. **Ley asociativa.** Se dice que un operador binario $*$ sobre un conjunto S es asociativo si:

$$(x * y) * z = x * (y * z) \text{ para todos } x, y, z \in S$$

3. **Ley conmutativa.** Se dice que un operador binario $*$ sobre un conjunto S es conmutativo si

$$x * y = y * x \text{ para todos } x, y \in S$$

4. **Elemento identidad.** Se dice que un conjunto S tiene un elemento de identidad respecto a una operación binaria $*$ sobre S si existe un elemento $e \in S$ con la propiedad

$$e * x = x * e = x \text{ para todos } x \in S$$

5. **Inverso.** Se dice que un conjunto S , que tiene el elemento de identidad e respecto a un operador $*$, tiene un inverso si, para todo $x \in S$, existe un elemento $y \in S$ tal que

$$x * y = e$$

6. **Ley distributiva.** Si $*$ y \cdot son dos operadores binarios sobre un conjunto S , decimos que $*$ es distributivo sobre \cdot si

$$x * (y \cdot z) = (x * y) \cdot (x * z)$$

2.2 Definición axiomática del álgebra booleana

En 1854, **George Boole** introdujo un tratamiento sistemático de la lógica. En 1938 **C. E. Shannon** introdujo un álgebra booleana de dos valores también llamada **álgebra de conmutación**.

Este álgebra es una estructura algebraica definida por un conjunto de elementos B , junto con dos operadores binarios, $+$ y \cdot , y seis postulados que introdujo **Huntington**.

1. Cerradura

Cerradura respecto al operador $+$.

Cerradura respecto al operador \cdot .

2. Elemento identidad

Elemento identidad con respecto a $+$, designado por 0: $x + 0 = 0 + x = x$.

Elemento identidad con respecto a \cdot , designado por 1, $x \cdot 1 = 1 \cdot x = x$.

3. Conmutativa

Conmutativa respecto a $+$: $x + y = y + x$.

Conmutativa respecto a \cdot : $x \cdot y = y \cdot x$.

4. Distributiva

\cdot es distributivo sobre $+$: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.

$+$ es distributivo sobre \cdot : $x + (y \cdot z) = (x + y) \cdot (x + z)$.

5. Complemento

Para cada elemento $x \in B$, existe un elemento un elemento $x' \in B$, tal que $x + x' = 1$ y $x \cdot x' = 0$.

6. Dualidad

Existen al menos dos elementos $x, y \in B$, tales que $x \neq y$.

2.3 Funciones booleanas

El álgebra booleana se ocupa de variables binarias y operaciones lógicas. Una función booleana, es descrita por una expresión algebraica que consta de variables binarias, contantes 0 y 1, y símbolos lógicos de operación. Por ejemplo:

$$F_1 = x + y'z$$

La función F_1 es igual a 1 si x es igual a 1 o si tanto y' como z son iguales a 1. En los demás casos, F_1 es igual a 0.

Se puede representar una función booleana en una **tabla de verdad**. Una tabla de verdad es una lista de *combinaciones* de unos y ceros asignados a las variables binarias y una columna que muestra el valor de la función para cada combinación. El número de filas de la tabla es de 2^n , donde n es el número de variables de la función. Contando de 0 hasta $2^n - 1$.

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 3: Tabla de verdad de $F_1 = x + y'z$

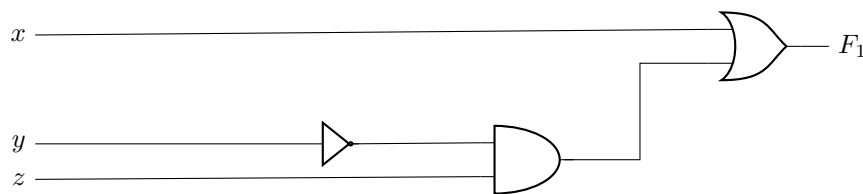


Figure 1: Circuito lógico de $F_1 = x + y'z$

Una función booleana se puede transformar de una expresión algebraica a un diagrama de circuitos hecho con compuertas lógicas. Solo hay una forma de representar una función booleana en una tabla de verdad. Sin embargo, la función en forma algebraica, puede expresarse de varias maneras. Manipulando una expresión booleana se puede obtener una expresión más simple de la misma función y así reducir el número de compuertas lógicas del circuito.

Consideremos ahora esta función booleana y su posible simplificación:

$$\begin{aligned} F_2 &= x'y'z + x'yz + xy' + xy' \\ &= x'z(y' + y) \\ &= x'z + xy' \end{aligned}$$

La función de tres terminos y ocho literales se reduce a únicamente dos términos y cuatro literales. Ambas realizan la misma función, pero es preferible la forma simplificada porque requiere menos compuertas lógicas.

2.3.1 Manipulación algebraica

Se define una literal como una sola variable dentro de un término. Si se reduce el número de términos, el número de literales, o ambas, en una expresión booleana, podría obtenerse un circuito más simple. Las funciones de hasta cinco variables se pueden simplificar con el método del mapa. Se describirá más adelante.

2.3.2 Complemento de una función

El complemento de una función F es F' , se obtiene intercambiando los ceros por unos y unos por ceros en el valor de F . Esto se puede deducir algebraicamente usando el teorema de **DeMorgan**. Además este se puede extender a tres o más variables. Así:

$$\begin{aligned} (A + B + C)' &= (A + x)' \\ &= A'x' \\ &= A'(B + C)' \\ &= A'(B'C') \\ &= A'B'C' \end{aligned}$$

El teorema de DeMorgan se puede generalizar de la siguiente manera:

$$\begin{aligned} (A + B + C + D + \dots + F)' &= A'B'C'D'\dots F' \\ (ABCD\dots F)' &= A' + B' + C' + D' + \dots + F' \end{aligned}$$

Otro procedimiento más sencillo para obtener el complemento de una función consiste en obtener el dual de la función y complementar cada literal. Esto es en consecuencia del teorema de DeMorgan. El dual de una función se obtiene intercambiando los operadores **AND** y **OR**, y unos y ceros. Ejemplo:

$$F_1 = x'yz' + x'y'z$$

El dual de F_1 es $(x' + y + z')(x' + y' + z)$

Complementando cada literal: $(x + y' + z)(x + y + z') = F'_1$

2.4 Formas canónicas y estándar

x	y	z	Minitérminos		Maxitérminos	
			Términos	Designación	Términos	Designación
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Table 4: Minitérminos y maxitérminos para tres variables binarias

Una variable binaria podría aparecer en su forma normal (x) o en su forma complementada (x'). Ahora si se considera dos variables binarias x y y que se combinan con una operación AND. Se pueden obtener cuatro combinaciones posibles: $x'y'$, $x'y$, xy' y xy . Cada uno de estos cuatro términos AND es un *minitérmino*, o *producto estándar*. De igual manera se puede combinar n variables para formar 2^n minitérminos. Se enumeran del 0 a $2^n - 1$. Cada minitérmino se obtiene de un término AND de las n variables, se designa un símbolo m_j , donde j denota el equivalente decimal del número binario que representa el minitérmino.

Asimismo, n variables que forman un término OR, llamadas *maxitérmino* o *suma estándar*. Cabe decir que cada maxitérmino es el complemento de su minitérmino correspondiente y viceversa. Así, expresar las combinaciones 001, 100 y 111 como $x'y'z'$, $xy'z'$ y xyz , respectivamente. Puesto que cada uno de estos minitérminos hace que $f_1 = 1$, se tiene:

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

2.4.1 Forma canónica: suma de minitérminos

Esto ilustra una propiedad importante del álgebra booleana: cualquier función booleana se puede expresar como *una suma de minitérminos*. Ahora podemos hacer lo mismo pero con maxitérminos, de modo que el complemento de f_2 se lee:

$$f'_2 = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

Ejemplo de suma de minitérminos:

Expresar la función booleana $F = A + B'C$ como la suma de minitérminos. Por lo tanto:

$$A = A(B + B') = AB + AB'$$

Aún le falta una variable:

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

Al segundo término, $B'C$, le falta una variable:

$$B'C = B'C(A + A') = AB'C + A'B'C$$

Al combinar todo se tiene:

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C \end{aligned}$$

Como $AB'C$ aparece dos veces, se puede simplificar:

$$\begin{aligned} F &= ABC + ABC' + AB'C + AB'C' + A'B'C \\ &= A'B'C + AB'C' + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

En ocasiones conviene expresar la función booleana, de la siguiente notación:

$$F(A, B, C) = \sum(1, 4, 5, 6, 7)$$

2.4.2 Forma canónica: producto de maxitérminos

Si obtenemos el complemento de f'_2 , se obtiene f_2 :

$$\begin{aligned} f_2 &= (x + y + z)(x + y' + z)(x + y' + z')(x' + y + z')(x' + y' + z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \end{aligned}$$

Este ejemplo ilustra la segunda propiedad del álgebra booleana, cualquier función booleana se puede expresar como un *producto de maxitérminos*. Se dice que las funciones booleanas expresadas como suma de minitérminos o producto de maxitérminos están en **forma canónica**.

Ejemplo de producto de maxitérminos:

Expresar la función booleana $F = xy + x'z$ en forma de producto de maxitérminos. Por lo tanto:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

La función tiene tres variables, x , y y z . A cada término OR le falta una variable; por tanto:

$$\begin{aligned}
x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\
x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\
y + z &= y + z + xx' = (x + y + z)(x' + y + z)
\end{aligned}$$

Después de combinar todos los términos y eliminar los que se repiten; se tiene:

$$\begin{aligned}
F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\
&= M_0 \cdot M_2 \cdot M_4 \cdot M_5
\end{aligned}$$

Una forma cómoda de expresar esta función es:

$$F(x, y, z) = \prod(0, 2, 4, 5)$$

2.5 Conversión entre formas canónicas