

Lógica y álgebra de Boole

Operadores booleanos y tablas de verdad

M. Antònia Huertas Sánchez

PID_00149518



Universitat Oberta
de Catalunya

www.uoc.edu

Índice

Introducción	5
Objetivos	6
1. Lógica booleana	7
1.1. Origen y objeto de la lógica booleana	7
1.2. Lenguaje y semántica de la lógica booleana	8
1.3. Operadores booleanos	9
1.3.1. Operadores básicos	9
1.3.2. Tablas de verdad	10
1.3.3. Otros operadores	12
1.3.4. Relaciones entre operadores	15
2. Lógica booleana e informática	19
2.1. Circuitos lógicos	19
2.2. Representación gráfica de circuitos lógicos	20
2.3. Otras aplicaciones	22
2.3.1. Criptografía	22
2.3.2. Programación	23
3. Álgebra de Boole	24
3.1. Definición de álgebra de Boole	24
3.2. Propiedades de las álgebras de Boole	25
3.3. Ejemplos de álgebras de Boole	26
Resumen	28
Ejercicios de autoevaluación	29
Solucionario	30
Glosario	36
Bibliografía	36

Introducción

La lógica booleana, tema de este módulo didáctico, es la lógica más simple con aplicación a la computación. Representa una inmejorable introducción a la lógica formal como herramienta para la representación formal de la información y la resolución de problemas. Detrás de la sencillez de los conceptos que utiliza está la potencia del rigor de las matemáticas que caracteriza a la lógica formal desde su comienzo. El tema central del módulo es la representación y el razonamiento con información que involucra sólo dos valores para todas las variables en juego.

En este módulo descubriréis conceptos teóricos fundamentales y la necesidad de formalización y matematización de los fundamentos de la ciencia de la computación. Además, contiene el germen de la manera como las ingenieras e ingenieros deben abordar los problemas en algunas de las especialidades de la informática.

Una de las características especiales de la lógica booleana es el uso de notaciones y símbolos diferentes para los mismos conceptos. Esto es consecuencia de la aplicabilidad de esta lógica a los más diversos contextos, hecho que ha generado notaciones específicas para cada uno de ellos. Entre las aplicaciones de esta lógica nos detendremos especialmente en la fundamentación de los circuitos digitales.

Se presenta también la estructura matemática abstracta que se construye a partir de la lógica booleana y que se conoce con el nombre de álgebra de Boole. Veréis cómo se generaliza un concepto y cómo esta generalización se puede aplicar a ámbitos nuevos que van más allá de la lógica.

Encontraréis muchos ejemplos, que es importante que leáis e intentéis entender. Al final del módulo hay ejercicios de autoevaluación para que podáis comprobar el nivel de comprensión de los conceptos y técnicas fundamentales del módulo.

Objetivos

En los recursos docentes facilitados en este módulo, encontraréis los conceptos, técnicas y herramientas necesarias para alcanzar los objetivos siguientes:

1. Comprender la necesidad de formalizar los conceptos para manipularlos con rigor.
2. Saber manipular algebraicamente las variables y los operadores booleanos y formar tablas de verdad.
3. Saber cómo expresar en la lógica booleana problemas susceptibles de formalizarse con su lenguaje.
4. Entender que la lógica no se ocupa del significado concreto de las expresiones y comprender el concepto de valor de significado.
5. Entender la aplicación de la lógica booleana a los circuitos digitales.
6. Comprender el concepto abstracto de álgebra de Boole y la diferencia con el de lógica booleana.

1. Lógica booleana

1.1. Origen y objeto de la lógica booleana

Su nombre deriva de **George Boole** (1815–1864), matemático británico que formuló el sistema del *Álgebra de la lógica* en 1847, en un libro de muy pocas páginas (*El análisis matemático de la lógica*) pero que significó el fin de la lógica aristotélica y el comienzo de la **lógica formal matemática** contemporánea. Boole tuvo la originalidad, en ese momento histórico, de utilizar las técnicas algebraicas para tratar expresiones de la lógica. El sistema así resultante se parece más a un sistema algebraico donde se definen unas operaciones sobre unas variables abstractas que tienen que cumplir unas propiedades y no otras (pensad en el caso del álgebra de las fracciones, donde se definen las operaciones de suma, resta, multiplicación y división y deben cumplir unas determinadas propiedades).


El concepto más general de **álgebra de Boole** aparece en 1860, en trabajos de William Jevons y Charles Sanders Peirce. Un álgebra de Boole es una **estructura algebraica** (esto es, definida a partir de unos elementos o variables y de operaciones con esos elementos) y que es axiomática (es decir, que tiene que cumplir unas determinadas propiedades que caracterizan a esas operaciones).

El objeto de estudio de la lógica, en general, es el estudio y la **representación del razonamiento correcto**. La lógica anterior a George Boole expresaba los razonamientos en un lenguaje natural, el mismo que vehicula la comunicación y los razonamientos cotidianos. Pero tanto la gran capacidad expresiva del lenguaje natural como la ambigüedad de muchas de sus expresiones (pueden tener un significado diferente dependiendo del contexto) no lo hacen la herramienta ideal para el razonamiento lógico riguroso. La lógica contemporánea recurre al uso de **lenguajes formales** específicos para ser más eficaz en la representación del razonamiento formal.

Entenderemos un **razonamiento formal** como una secuencia de información formulada en un **lenguaje formal** (sin ambigüedad) en dos partes diferenciadas. Así, en un razonamiento hay dos partes bien diferentes: una primera parte (**premisa del razonamiento**) que es la formulación de conocimiento o información que se acepta como válida y una segunda parte (**consecuencia del razonamiento**) que es la formulación de conocimiento cuya validez se sigue *lógicamente* de la validez de la primera parte de una manera *automática*.

Un elemento fundamental de la lógica es el **lenguaje formal** sin ambigüedad que se usa para formalizar el conocimiento o información. El segundo

elemento fundamental de toda lógica es la definición de lo que se considera un **razonamiento válido**, es decir, de cómo se deben seguir las consecuencias a partir de las premisas en un razonamiento válido o correcto de esa lógica. El tercer elemento imprescindible en una lógica es la **semántica**, que consiste en la definición rigurosa de cómo se pueden interpretar las expresiones del lenguaje formal para asignarles valores de significado (lo más habitual es asignarles uno de los dos valores verdadero o falso).

Observad que la semántica de la lógica no se ocupa del significado de la expresión formal en un contexto determinado (esto es, de si la expresión se puede referir a personas, circuitos o programas) sino que la semántica de la lógica se ocupa de definir cuáles serán **los valores lógicos de significado** (en la lógica clásica son dos: verdadero y falso, pero hay otras lógicas que tienen otros valores, como la *lógica fuzzy* que tiene infinitos) y la forma cómo se corresponden esos valores lógicos con las expresiones y con los razonamientos. 

Una lógica se compone de tres elementos característicos:


- a) Lenguaje formal
- b) Razonamiento válido formal
- c) Semántica formal

La **lógica booleana** es la más sencilla de las lógicas formales. Está inspirada en el sistema de Boole y, curiosamente, una de sus aplicaciones más importantes se da en la informática, en particular en la fundamentación teórica de los circuitos digitales. Es además componente fundamental de la lógica de enunciados y de la lógica de predicados (los sistemas de lógica formal más conocidos).

A continuación presentamos el lenguaje y la semántica de la lógica booleana pero no el concepto de razonamiento válido. El concepto de razonamiento válido de la lógica de enunciados es una generalización del que se podría dar en la lógica booleana.

1.2. Lenguaje y semántica de la lógica booleana

El lenguaje formal de la lógica booleana es muy sencillo, ya que las únicas expresiones formales son **las variables del lenguaje**, todas de una misma tipología. Usaremos los símbolos formales A, B, C, \dots , que no tienen un significado en particular, para representar formalmente cualquier expresión de la lógica booleana.

Lo que realmente caracteriza la lógica booleana es que cualquier expresión o variable sólo puede interpretarse con uno de los **dos valores de significado posibles**, que suelen representarse por 1 (equivalente al valor de verdad **verdadero**) y 0 (equivalente al valor de verdad **falso**). 

Semántica booleana

En lógica booleana no interesa el significado concreto de una expresión, sino que solamente interesa su evaluación como 1 o 0.

Los valores que toman las variables en la **semántica de la lógica booleana** son 1 y 0 (verdadero y falso).

Para indicar que una expresión A **se evalúa como 1**, escribiremos $A = 1$ y para indicar que **se evalúa como 0**, escribiremos $A = 0$.

1.3. Operadores booleanos

Seguidamente veremos los operadores básicos, las tablas de verdad, otros operadores y las relaciones entre operadores.


1.3.1. Operadores básicos

Si A y B son expresiones o variables booleanas, se pueden generar nuevas expresiones a partir de ellas combinándolas con los llamados operadores booleanos.

Los operadores booleanos básicos son los que aparecen en la siguiente tabla:

Operador booleano (nombre)	Símbolo	Símbolo alternativo
Producto lógico o conjunción	\cdot	\wedge <i>AND</i>
Suma lógica o disyunción	$+$	\vee <i>OR</i>
Complementación o negación	\sim	\neg <i>NOT</i>


En la tabla anterior se han privilegiado los símbolos originales de George Boole, pero como se ve en la columna de símbolos alternativos, hay maneras diferentes de escribir los operadores booleanos básicos. Hemos puesto como representaciones alternativas las utilizadas en dos de las aplicaciones más importantes de la lógica booleana: los **operadores de conjunción, disyunción y**

La lógica de enunciados se presenta en el módulo "Lógica de enunciados" de esta asignatura. 

negación en el contexto de la **lógica de enunciados** (\wedge, \vee, \neg) y los símbolos de los mismos operadores utilizados **en el contexto de la lógica computacional** (**AND**, **OR** y **NOT**).

La manera como los operadores básicos forman expresiones complejas a partir de otras más simples A, B se muestra en esta otra tabla:

Expresión	Nombre	Significado
$A \cdot B$	"A y B"	$\left\{ \begin{array}{ll} A \cdot B = 1 & \text{si } A = B = 1 \\ A \cdot B = 0 & \text{en otro caso} \end{array} \right.$
$A + B$	"A o B"	$\left\{ \begin{array}{ll} A + B = 1 & \text{si } A = 1 \text{ o } B = 1 \\ A + B = 0 & \text{en otro caso} \end{array} \right.$
$\sim A$	"no A"	$\left\{ \begin{array}{ll} \sim A = 1 & \text{si } A = 0 \\ \sim A = 0 & \text{si } A = 1 \end{array} \right.$

Con los operadores, por tanto, se forman expresiones complejas a partir de otras expresiones más simples. Pero lo que realmente es importante en el sistema inventado por Boole es que el valor que toma la expresión compleja (1 o 0) depende sólo del valor que toma cada una de las expresiones que la componen. El valor de una expresión compleja se puede *calcular* a partir de los valores de las expresiones más simples que la componen, con un algoritmo llamado **tabla de verdad** de la expresión. 

Para presentar el concepto general de tabla de verdad es necesario primero definir las tablas de verdad de cada operador.

1.3.2. Tablas de verdad

La tabla de verdad de un operador es la que recoge el valor de una expresión formada con ese único operador a partir de los posibles valores de las expresiones que lo componen.

A continuación se especifican las tablas de valores para los tres operadores básicos. En las columnas de las expresiones A y B aparecen todos los posibles pares de valores que pueden tomar. En la columna de la expresión compleja formada con el operador se recoge el valor que corresponde al par de valores de las expresiones más simples que hay en cada fila.

A	B	$A \cdot B$	A	B	$A + B$	A	$\sim A$
1	1	1	1	1	1	1	0
1	0	0	1	0	1	0	1
0	1	0	0	1	1		
0	0	0	0	0	0		

Interpretación de las tablas de verdad

Para entender cómo se debe interpretar la tabla de verdad de un operador tomemos, por ejemplo, la tabla del operador conjunción: cuando $A = 1$ y $B = 1$ el valor de la expresión compleja $A \cdot B = 1$ (es la primera fila de la tabla); respectivamente, cuando $A = 1$ y $B = 0$ el valor de la expresión compleja $A \cdot B = 0$ (es la segunda fila de la tabla); y así sucesivamente.

Observad que las tablas se corresponden con los valores de la expresión dada en la tabla de presentación de los operadores en el subapartado anterior:

- **Producto lógico o conjunción.** La expresión $A \cdot B$ toma el valor 1 si las expresiones A y B lo toman simultáneamente y toma el valor 0 en cualquier otro caso.
- **Suma lógica o disyunción.** La expresión $A + B$ toma el valor 1 si al menos una de las expresiones A o B toman el valor 1 y toma el valor 0 en cualquier otro caso.
- **Complementación o negación.** La expresión $\sim A$ toma el valor 1 si la expresión A toma el valor 0, y toma el valor 0 si la expresión A toma el valor 1.

Ejemplo de tablas de verdad de expresiones booleanas

a) $(\sim A) \cdot (A + B)$

Primero se construyen las tablas de verdad de $\sim A$ y de $A + B$, y a continuación la tabla de la conjunción de ambas.

A	B	$\sim A$	$A + B$	$(\sim A) \cdot (A + B)$
1	1	0	1	0
1	0	0	1	0
0	1	1	1	1
0	0	1	0	0

Expresión compleja

La tabla de verdad de una expresión compleja se construye a partir de las tablas de las subexpresiones que la componen.

b) $(A + B) \rightarrow (\sim C)$

A	B	C	A + B	$\sim C$	$(A + B) \rightarrow (\sim C)$
1	1	1	1	0	0
1	1	0	1	1	1
1	0	1	1	0	0
1	0	0	1	1	1
0	1	1	1	0	0
0	1	0	1	1	1
0	0	1	0	0	1
0	0	0	0	1	1

1.3.3. Otros operadores

Una pregunta inmediata es si los operadores booleanos básicos definidos anteriormente son los únicos. Y si no es así, ¿hay más operadores?

Veamos primero el caso de operadores que actúan sobre una única expresión. En la siguiente tabla se presentan todos los posibles operadores **unarios** (también llamados **monarios**), es decir, los que operan sobre una expresión.

Sólo hay cuatro posibles maneras de dar dos valores de salida (1 o 0) a partir de los valores 1,0 de entrada, que se muestran en la tabla siguiente, y eso condiciona el número de operadores unarios posibles, que es también cuatro:

A	o_1	o_2	o_3	o_4
1	1	1	0	0
0	1	0	1	0

En la tabla siguiente se puede leer el nombre de estos cuatro operadores. Como podéis ver solamente **la negación** (o_3) es un operador significativo y por ello tiene símbolos conocidos. El operador negación se usa en todas las aplicaciones de la lógica en la computación y la electrónica.

Operador	Nombre	Símbolo
o_1	constante 1	
o_2	identidad	
o_3	negación	$\sim \quad \neg \quad NOT$
o_4	constante 0	

Funciones lógicas

En el contexto de la computación los operadores booleanos reciben el nombre de **funciones lógicas**.

Estudiaremos ahora el caso de operadores que actúan sobre dos expresiones, llamados **operadores binarios**. En la siguiente tabla se muestran los 16 operadores posibles, a partir de todas las combinaciones posibles de valores que pueden tomar las dos expresiones simples sobre las que operan (A y B).

A	B	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}	o_{11}	o_{12}	o_{13}	o_{14}	o_{15}	o_{16}
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
0	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

La ordenación de los 16 operadores binarios no es significativa. En la tabla anterior se ha elegido una regla de llenado de la tabla para disponer las 16 posibilidades que consiste en escribir en la primera fila de la tabla ocho 1 seguidos de ocho 0, en la segunda fila alternar cuatro 1 con cuatro 0, en la tercera fila alternar dos 1 con dos 0 y en la cuarta fila alternar un 1 con un 0. De esta manera nos aseguramos que se han escrito todas las combinaciones posibles.

Sólo algunos de los 16 operadores binarios tienen significación y corresponden a operadores booleanos conocidos. Además, hay que tener siempre presente que existe diversidad de símbolos para un mismo operador, dependiendo del contexto donde se usa.

En la siguiente tabla se muestran los símbolos más utilizados para los operadores binarios más comunes en tres de los contextos más importantes, el de la lógica booleana, el de la lógica clásica (lógica de enunciados y lógica de predicados) y el de la computación.

Operador	Nombre	Booleana	Clásica	Computación
o_8	conjunción	\cdot	\wedge	<i>AND</i>
o_2	disyunción	$+$	\vee	<i>OR</i>
o_9	nand		\uparrow	<i>NAND</i>
o_{15}	nor		\downarrow	<i>NOR</i>
o_5	implicación	\supset	\longrightarrow	
o_7	equivalencia	\equiv	\longleftrightarrow	
o_{10}	disyunción exclusiva	\oplus	\oplus	\oplus

Observad que sólo hay signos de todos ellos para la lógica clásica (lógica de enunciados y lógica de predicados), que es el sistema lógico más importante que contiene los operadores de la lógica booleana.

A continuación se muestra la tabla de verdad de los operadores booleanos binarios más comunes, usando, en este caso, los símbolos propios de la lógica clásica (de enunciados y de predicados).

A	B	\wedge	\vee	\longrightarrow	\longleftrightarrow	\uparrow	\downarrow	\oplus
1	1	1	1	1	1	0	0	0
1	0	0	1	0	0	1	0	1
0	1	0	1	1	0	1	0	1
0	0	0	0	1	1	1	1	0

Observación

La tabla se debe leer como el valor de la expresión $A \wedge B$ a partir del valor de las expresiones de A y de B , el valor de $A \vee B$ a partir del valor de las expresiones de A y de B y así sucesivamente.

La implicación

La expresión $A \longrightarrow B$ se lee “ A implica B ”, donde A se llama **el antecedente** y B **el consecuente** de la expresión.

El significado de esta expresión es que si el antecedente es verdadero ($A = 1$) también lo es el consecuente ($B = 1$). Por ello, $A \longrightarrow B$ sólo es falsa (su valor es 0) cuando el antecedente es verdadero ($A = 1$) pero el consecuente es falso ($B = 0$).

Ejemplo de implicación

Si consideramos las expresiones $A = [2 + 2 = 5]$ y $B = [\text{la Tierra es plana}]$, ambos A y B son falsas ($A = 0$ y $B = 0$). Por tanto, $A \longrightarrow B$ es verdadero ($A \longrightarrow B = 1$). Así que la expresión $[2 + 2 = 5] \longrightarrow [\text{la Tierra es plana}]$ es verdadera.

Sin embargo, la expresión $[\text{la Tierra es plana}] \longrightarrow [2 + 2 = 4]$ es falsa, ya que $[2 + 2 = 4]$ es verdadera (tiene valor 1) y $[\text{la Tierra es plana}]$ es falsa (tiene valor 0).

Implicación

El valor de la implicación puede ser verdadero aunque el consecuente sea falso.

La disyunción exclusiva

La expresión $A \oplus B$ se lee “ A o B pero no ambas” e indica que o bien A o bien B tienen el valor 1, pero no simultáneamente.

Ejemplo de disyunción exclusiva

La frase “*Me inscribiré en la UOC el lunes o el martes*” corresponde con el significado del operador disyunción exclusiva, ya que no puedo inscribirme dos veces.

Operadores *nand* y *nor*

La expresión $A \uparrow B$ se lee “ A *nand* B ” e indica que A y B no tienen simultáneamente el valor 1.

La expresión $A \downarrow B$ se lee “ A *nor* B ” e indica que ni A ni B tienen el valor 1.

Podemos definir operadores terciarios, cuaternarios, etc., dependiendo del número de valores de entrada que se operan para dar un valor de salida. En el caso de operadores terciarios hay $4^3 = 64$ posibilidades, en el caso de los cuaternarios $4^4 = 256$ posibilidades. En general para operadores n -arios (con n valores de entrada) tenemos 4^n posibilidades.

En lo que nos interesa, nos concentramos en los operadores unarios y binarios, que son los más usados en la aplicación de la lógica booleana a la computación.

1.3.4. Relaciones entre operadores

De los 16 operadores binarios hay muchos que se pueden definir a partir de otros operadores binarios y unarios.

o_1 es la negación de o_{16} ya que sus columnas de valores respectivas son contrarias entre sí. Igualmente, ocurre que los operadores del o_9 al o_{15} son la negación, respectivamente, de los operadores del o_8 al o_2 . En particular, observad que el operativo *nor* (o_{15}) es la negación del operador disyunción (o_2) y que *nand* (o_9) es la negación del operador conjunción (o_8). Por tanto, de los 16 operadores binarios podemos quedarnos con los ocho primeros, ya que los otros ocho se pueden definir en función de éstos y del operador unario negación.

Por otro lado, o_4 y o_6 son operadores proyección, esto es $A o_4 B = A$ y $A o_6 B = B$, y el operador o_1 es un operador constante; por tanto, también estos tres son superfluos.

Con este sencillo análisis, el conjunto inicial de los 16 operadores binarios posibles es redundante, ya que podemos quedarnos solamente con los cinco operadores binarios o_2, o_3, o_5, o_7, o_8 y la negación. Además, entre éstos observamos que o_3 y o_5 son inversos entre sí ($A o_3 B = B o_5 A$) y que o_7 se expresa en función de o_5 y o_8 ($A o_7 B = (A o_5 B) o_8 (B o_5 A)$). Por tanto, con los tres operadores binarios disyunción, implicación y conjunción (o_2, o_5 y o_8 , respectivamente) y el operador unario negación es posible definir los 16 operadores binarios. En el caso de los operadores unarios, el operador negación junto a uno de los operadores constantes (por ejemplo el constante 1) permite definir los cuatro operadores unarios.

Recordad

$$\begin{aligned} A o_5 B &= A \rightarrow B \\ A o_7 B &= A \leftrightarrow B \\ A o_8 B &= A \wedge B \end{aligned}$$

Llamaremos conjunto de **operadores primitivos** a cualquier conjunto de operadores binarios y unarios que permite definir todos los operadores unarios y binarios de la lógica booleana.

Es fácil comprobar que son conjuntos de operadores primitivos los siguientes:

- a) El conjunto formado por los operadores negación y conjunción (\sim y \cdot).
- b) El conjunto formado por los operadores negación y disyunción (\sim y $+$).
- c) El conjunto formado por el operador *nand* o el operador *nor* (sólo uno de ellos).

Para demostrar lo anterior, es suficiente comprobar que los tres operadores binarios disyunción, implicación y conjunción, y el operador negación se pueden obtener a partir de los respectivos conjuntos de operadores primitivos:

- a) Disyunción, implicación, conjunción y negación se obtienen a partir de \sim y \cdot :

$$A + B = \sim((\sim A) \cdot (\sim B))$$

$$A \supset B = \sim(A \cdot (\sim B))$$

- b) Disyunción, implicación, conjunción y negación se obtienen a partir de \sim y $+$:

$$A \cdot B = \sim((\sim A) + (\sim B))$$

$$A \supset B = (\sim A) + B$$

- c) Basta comprobar que el operador *nand*, o el operador *nor* (sólo uno de ellos), es suficiente para definir los operadores disyunción y negación (que son un conjunto de operadores primitivos)

- Comprobamos para el operador *nand* (\uparrow , *NAND*)

$$A + B = (A \uparrow A) \uparrow (B \uparrow B)$$

$$A \cdot B = (A \uparrow B) \uparrow (A \uparrow B)$$

$$\sim A = A \uparrow A$$

Lo verificamos construyendo las tablas de verdad correspondientes, y comprobando que son iguales:

A	B	$A \uparrow A$	$B \uparrow B$	$(A \uparrow A) \uparrow (B \uparrow B)$	$A + B$
1	1	0	0	1	1
1	0	0	1	1	1
0	1	1	0	1	1
0	0	1	1	0	0

A	$A \uparrow A$	$\sim A$
1	0	0
0	1	1

- Comprobamos para el operador *nor* (\downarrow , NOR). Basta ver que *nor* se puede expresar con *nand* (\uparrow , NAND).

$$A \downarrow B = \sim(A + B) = \sim[(A \uparrow A) \uparrow (B \uparrow B)]$$

Ejercicios sobre operadores

1) Evaluad el valor de la expresión $(A + B) \cdot C$ en los casos siguientes:

a) $A = 0, B = 1, C = 1$; b) $A = 1, B = 0, C = 1$; c) $A = 1, B = 1, C = 0$

Para ello se construye la tabla de verdad para estos tres casos.

	A	B	C	$A + B$	$(A + B) \cdot C$
a)	0	1	1	1	1
b)	1	0	1	1	1
c)	1	1	0	1	0

2) Expresa $(A \supset B) + (\sim A \supset C)$ en función de los operadores \sim y \cdot .

Como $A \supset B = \sim(A \cdot (\sim B)) = X$ y

$\sim A \supset C = \sim((\sim A) \cdot (\sim C)) = Y$, entonces

$(A \supset B) + (\sim A \supset C) = X + Y = \sim((\sim X) \cdot (\sim Y)) =$

$\sim((\sim(\sim(A \cdot (\sim B)))) \cdot (\sim(\sim((\sim A) \cdot (\sim C))))))$

3) Expresa $(A \supset B) \cdot C$ en función de los operadores \sim y $+$

$(A \supset B) \cdot C = (\sim A + B) \cdot C = \sim(\sim(\sim A + B) + (\sim C))$

4) Expresa $(A \cdot B) + C$ en función del operador \uparrow

Primero ponemos $A \cdot B$ en función de $+$ y \sim

$A \cdot B = \sim((\sim A) + (\sim B))$

entonces

$$(A \cdot B) + C = \sim((\sim A) + (\sim B)) + C$$

Como $A + B = (A \uparrow A) \uparrow (B \uparrow B)$ sustituimos $+$ por \uparrow

$$(\sim A) + (\sim B) = ((\sim A) \uparrow (\sim A)) \uparrow ((\sim B) \uparrow (\sim B))$$

y entonces

$$(A \cdot B) + C = \sim(((\sim A) \uparrow (\sim A)) \uparrow ((\sim B) \uparrow (\sim B))) + C$$

Si llamamos

$$X = \sim(((\sim A) \uparrow (\sim A)) \uparrow ((\sim B) \uparrow (\sim B)))$$

tenemos

$$(A \cdot B) + C = X + C = (X \uparrow X) \uparrow (C \uparrow C)$$

Por otro lado,

$\sim A = A \uparrow A$ y podemos sustituir \sim por \uparrow

y, por tanto,

$$X = [(((A \uparrow A) \uparrow (A \uparrow A)) \uparrow ((B \uparrow B) \uparrow (B \uparrow B))) \uparrow [(((A \uparrow A) \uparrow (A \uparrow A)) \uparrow ((B \uparrow B) \uparrow (B \uparrow B)))]$$

Finalmente, sustituyendo X en $(A \cdot B) + C = (X \uparrow X) \uparrow (C \uparrow C)$ obtenemos la expresión de $(A \cdot B) + C$ sólo con el operador *nand* (\uparrow , *NAND*)


La extensión de la fórmula resultante es tan grande que no cabe en una sólo línea del espacio físico de este folio. Esta dificultad es un inconveniente habitual de la escritura de una expresión sólo con el operador *nand* o el operador *nor*. En el apartado siguiente se mostrará una manera más eficaz de obtener (gráficamente) la expresión con estos operadores.

2. Lógica booleana e informática

2.1. Circuitos lógicos

Una de las aplicaciones más importantes de la lógica booleana son los **computadores digitales**.

Un circuito electrónico digital es un sistema formado por **señales de entrada**, cada una correspondiente a un cable, una serie de **dispositivos electrónicos** que operan sobre las señales de entrada y que dan como resultado una **señal de salida**, correspondiente a un cable. Los cables que forman los circuitos de los computadores digitales pueden estar en dos valores de tensión (baja o alta) y estos dos valores se pueden identificar con los valores 1 y 0. Cuando las señales de entrada o de salida sólo pueden tomar dos valores, se dice que son **señales lógicas o binarias**. Las operaciones que se pueden hacer sobre ellas corresponden a los operadores booleanos, que en el contexto de los circuitos lógicos reciben el nombre de **funciones lógicas**.

La característica booleana fundamental es la de que las señales sólo puedan tomar dos valores, porque entonces si no toma uno de ellos, automáticamente podemos inferir que toma el otro. 

Los circuitos lógicos, por tanto, se basan en la propiedad de que recibiendo un número de señales como entrada (1 o 0) operan sobre ellas para producir una señal como salida (1 o 0). Esta señal de salida se puede representar como una expresión booleana compleja formada a partir de las expresiones simples de las señales de entrada, usando operadores booleanos.

Ejemplo de representación con lógica booleana

Imaginemos que un circuito electrónico digital consta de cuatro cables de entrada A, B, C, D y supongamos que para que funcione correctamente se debe cumplir que los cables A y B no pueden estar simultáneamente en alta tensión, y que los cables C y D deben estar los dos a la vez en alta tensión o los dos a la vez en baja tensión. Supongamos, además, que nuestro problema consiste en controlar la tensión de A, B, C, D en cada momento y saber si el dispositivo funciona correctamente.

Las dos condiciones de funcionamiento exigidas en este ejemplo se pueden representar con la expresión booleana:

- 1) A y B no están simultáneamente en alta tensión: $\sim (A \cdot B)$ (observad que expreso que A está en alta tensión con A , entonces debo expresar que A está en baja tensión con el valor contrario, es decir con su negación $\sim A$)
- 2) C y D están a la vez en alta o a la vez en baja tensión: $(C \cdot D) + (\sim C \cdot \sim D)$

La condición de funcionamiento correcto del dispositivo es la conjunción de las dos anteriores y se puede expresar: $\sim(A \cdot B) \cdot [(C \cdot D) + (\sim C \cdot \sim D)]$

Para saber la correspondencia entre el valor de la señal de los cables cables A, B, C, D y de la expresión que representa el funcionamiento correcto, se construye la tabla de verdad de esta expresión.

Para ello, a partir de los posibles valores de A, B, C, D , calculamos primero el valor de la expresión $A \cdot B$ y a partir de éste el de $\sim(A \cdot B)$. Por otro lado, se calcula sucesivamente el valor de $C \cdot D$ y $\sim C \cdot \sim D$ y $X = (C \cdot D) + (\sim C \cdot \sim D)$. Finalmente, a partir de la columna de $\sim(A \cdot B)$ y de X se calcula el valor de $Y = \sim(A \cdot B) \cdot X$, que es su conjunción:

A	B	C	D	$A \cdot B$	$\sim(A \cdot B)$	$C \cdot D$	$\sim C \cdot \sim D$	X	Y
1	1	1	1	1	0	1	0	1	0
1	1	1	0	1	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0
1	1	0	0	1	0	0	1	1	0
1	0	1	1	0	1	1	0	1	1
1	0	1	0	0	1	0	0	0	0
1	0	0	1	0	1	0	0	0	0
1	0	0	0	0	1	0	1	1	1
0	1	1	1	0	1	1	0	1	1
0	1	1	0	0	1	0	0	0	0
0	1	0	1	0	1	0	0	0	0
0	1	0	0	0	1	0	1	1	1
0	0	1	1	0	1	1	0	1	1
0	0	1	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	0	0	1	0	1	1	1

Con esta tabla es fácil resolver el problema de si el dispositivo funciona o no correctamente a partir de las posiciones de los valores de entrantes.

Por ejemplo, si la persona responsable del dispositivo observa que $A = 1, B = 0, C = 1$ y $D = 1$, mirando la fila correspondiente (fila cinco) de la tabla anterior sabrá que el dispositivo está funcionando correctamente ($Y = 1$).

2.2. Representación gráfica de circuitos lógicos

En la figura 1 podéis ver cómo se representan los operadores básicos como puertas lógicas.

Ejemplo de puertas lógicas

En la figura 2 se representa la expresión $(\sim A + B) + \sim(C \cdot D)$

Lo habitual es representar los circuitos con las llamadas **puertas universales**. Las más comunes son $NAND = NOT(AND)$ y $NOR = NOT(OR)$. En la figura 3 se representan las puertas lógicas $NAND$ y NOR .

Paréntesis

Obsérvese que utilizamos los paréntesis para clarificar las expresiones, sin que aporten ningún valor expresivo nuevo.

Puerta lógica

Es donde la señal de entrada produce la señal de salida.

Figura 1

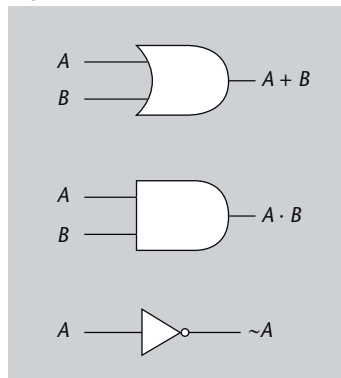


Figura 2

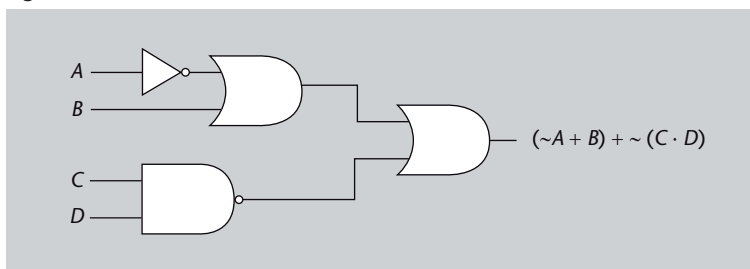
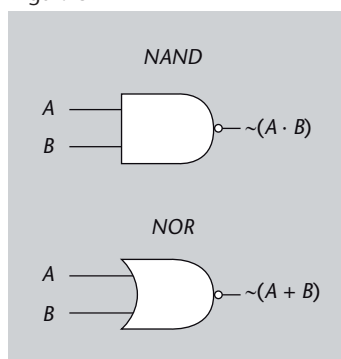


Figura 3

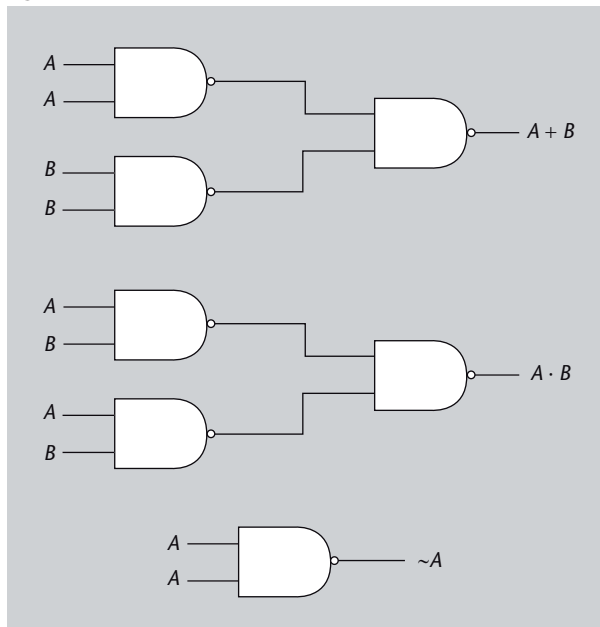


Lo interesante de estos dos operadores es que cualquier circuito se puede modelar usando un solo tipo de ellos.

En la figura 4 se ve cómo los operadores básicos disyunción, conjunción y negación se pueden modelar con puertas *NAND*.

En el diseño de *hardware* digital, las puertas lógicas normalmente implementadas son negación, conjunción y disyunción. Las negaciones de éstas dos últimas, *nand* y *nor*, son también importantes porque sólo implementando una de ellas permiten la posterior implementación de todas las demás. En particular, $\sim(A \cdot B)$ se puede sustituir por la puerta individual *NAND*.

Figura 4

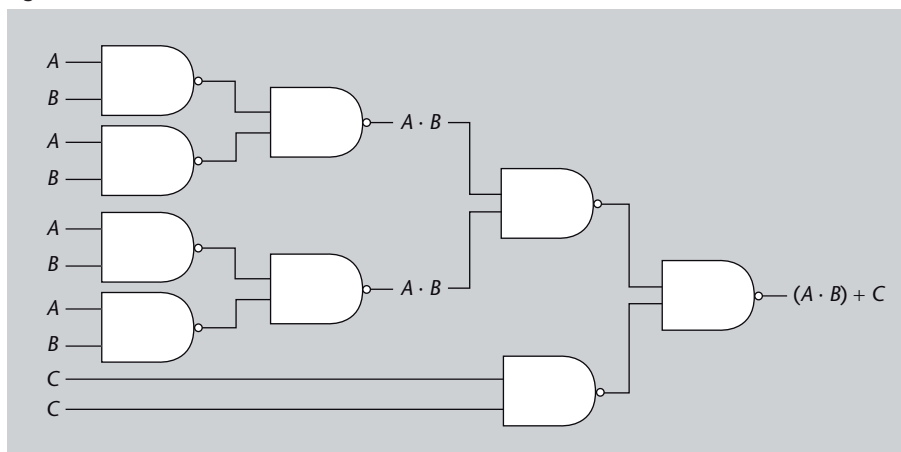


Ejemplo de modelo con puertas NAND

Ya vimos cómo expresar $(A \cdot B) + C$ en función del operador \uparrow . El resultado era una expresión muy compleja. Veremos ahora en la figura 5, cómo se puede modelar esa expresión usando la puerta NAND.

Ved la expresión $(A \cdot B) + C$ en función del operador \uparrow en el ejercicio 4 del subapartado 1.3.4.

Figura 5



2.3. Otras aplicaciones

2.3.1. Criptografía

Los operadores binarios \longleftrightarrow y \oplus son negación uno de otro, es decir:

$$A \oplus B = \neg(A \longleftrightarrow B)$$

$$A \longleftrightarrow B = \neg(A \oplus B)$$

Podéis ver las columnas de a_7 y a_{10} en la tabla de verdad de operadores binarios del subapartado 1.3.3.

Son también sus propios inversos, es decir:

$$((A \longleftrightarrow B) \longleftrightarrow B) = A$$

$$((A \oplus B) \oplus B) = A.$$

Esta propiedad puede usarse en circuitos y programas para codificar en criptografía. Un mensaje codificado se crea con una operación entre los datos y el código. Si en la expresión:

$$\text{MensajeCodificado} = \text{Datos} \oplus \text{Código}$$

Los datos se pueden recuperar usando el operador disyunción exclusiva sobre *MensajeCodificado* ya que:

$$\text{MensajeCodificado} \oplus \text{Código} = (\text{Datos} \oplus \text{Código}) \oplus \text{Código} = \text{Datos}$$

2.3.2. Programación

Una de las aplicaciones de la lógica booleana son los lenguajes de programación. Muchos de los algoritmos contienen expresiones del tipo *if* y *while*.

Otra importante aplicación está en los sistemas de recuperación de la información como los buscadores de Internet, en los que se puede introducir una *query* como:

$$[\text{Booleana}] \text{ AND } [\text{lógica}] \text{ AND NOT } [\text{proposicional}] \text{ OR } [\text{predicados}]$$

que significa: buscar todos los documentos en los que aparezcan los términos *Booleana* y *lógica*, pero no aparezcan *proposicional* o *predicados*, si estoy buscando documentos sobre lógica booleana pero que no traten de lógica proposicional o de lógica de predicados.

3. Álgebra de Boole

3.1. Definición de álgebra de Boole

La manipulación de las expresiones booleanas, sus reglas y propiedades definen una estructura matemática abstracta que se puede aplicar a otros contextos, tanto de la matemática como de la informática. Esa estructura abstracta se llama álgebra de Boole y, aunque está inspirada en la lógica booleanas, es más compleja y abstracta.

El concepto de álgebra de Boole es una generalización de la lógica booleana. La generalización consiste en identificar los elementos característicos de la estructura que subyace a la lógica booleana: los dos valores (uno opuesto al otro), los operadores booleanos primitivos y sus tablas de verdad. Por tanto, cualquier generalización de la lógica booleana, y en particular la llamada álgebra de Boole, habrá de contener dos valores significativos equivalentes a 0 y a 1; y unas operaciones equivalentes a un conjunto primitivo de operadores booleanos que operando sobre los valores 0 y 1 produzcan el mismo resultado que en las tablas de verdad booleanas.

Así, un álgebra de Boole es una estructura matemática formada por un conjunto A que contiene, en particular, dos elementos diferentes, que vamos a llamar *cero* y *uno* (0 y 1, aunque podrían escribirse con otros símbolos), una operación unaria (vamos a simbolizar \sim) y dos operaciones binarias (que vamos a simbolizar $+$ y \cdot) y que han de cumplir una serie de propiedades o axiomas. Obsérvese que el conjunto A puede contener más elementos además del 0 y 1.

Se llama álgebra de Boole a la estructura matemática formada por un conjunto A , que tiene al menos dos elementos diferentes (0 y 1) y sobre el que se definen tres operaciones ($+$, \cdot , \sim) que cumplen las propiedades 1) a 5) siguientes, donde x , y , z representan elementos cualesquiera del conjunto A :

1) Conmutativa:

$$x + y = y + x$$

$$x \cdot y = y \cdot x$$

2) Asociativa:

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

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

3) Distributiva:

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

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

4) Identidad. Se dice que 0 es elemento neutro de + y que 1 es el elemento neutro de ·


$$x + 0 = x$$

$$x \cdot 1 = x$$

5) Complementación. Se dice que $\sim x$ es el complementario de x

$$x + (\sim x) = 1$$

$$x \cdot (\sim x) = 0$$

Este álgebra de Boole se representa con la expresión $(A, 0, 1, +, \cdot, \sim)$. Es un ejercicio comprobar que cuando las operaciones del álgebra de Boole se definen como los operadores booleanos de la lógica booleana (sus tablas de valores) se cumplen las propiedades anteriores. Por tanto, la lógica booleana es un álgebra de Boole (en este caso el conjunto A sólo contiene los elementos 0 y 1). 

Lógica booleana

Es un álgebra de Boole con sólo dos elementos (0 y 1).

Una **expresión algebraica** en un álgebra de Boole es una expresión construida a partir de elementos del conjunto A del álgebra y usando las tres operaciones básicas.

3.2. Propiedades de las álgebras de Boole

Además de estas propiedades fundamentales o axiomas de las álgebras de Boole, se cumplen una serie de propiedades o teoremas significativos y propios de las álgebras de Boole.

Si x, y, z son variables cualesquiera de un Álgebra de Boole, se cumplen los siguientes teoremas:

1) Idempotencia

$$x + x = x$$

$$x \cdot x = x$$

2) Doble negación

$$\sim\sim x = x$$

3) Absorción

$$x + (x \cdot y) = x$$

$$x \cdot (x + y) = x$$

4) Dominancia


$$x + 1 = 1$$

$$x \cdot 0 = 0$$

5) Leyes de De Morgan

$$\sim(x + y) = (\sim x) \cdot (\sim y)$$

$$\sim(x \cdot y) = (\sim x) + (\sim y)$$

Estas propiedades se cumplen para los operadores de la lógica booleana y, por tanto, se pueden utilizar para simplificar expresiones y operar entre expresiones con más facilidad. Pensad por un momento en la propiedad de la doble negación que permite eliminar de una expresión dos negaciones seguidas, o las leyes de De Morgan que permiten intercambiar los operadores conjunción y disyunción. 

3.3. Ejemplos de álgebras de Boole

1) En lógica

El Álgebra de Boole más simple es la que tiene sólo dos elementos, 0 y 1, y es la que coincide con las operaciones básicas de la **lógica booleana**.

A	B	$A \cdot B$	A	B	$A + B$	A	$\sim A$
1	1	1	1	1	1	1	0
1	0	0	1	0	1	0	1
0	1	0	0	1	1		
0	0	0	0	0	0		

Es un ejercicio trivial comprobar que cumple todas las propiedades de un álgebra de Boole, haciendo la tabla de valor para las expresiones que aparecen en las diferentes propiedades.


Esta álgebra de Boole tiene aplicaciones importantes:

- En el diseño de circuitos digitales, interpretando 0 y 1 como los dos diferentes estados de un bit en un circuito.

$(\{0,1\}, AND, OR, NOT)$ es un álgebra de Boole.

- En la lógica, interpretando el elemento 0 como *falso* y el elemento 1 como *verdadero*.


$(\{0,1\}, \wedge, \vee, \neg)$ es un álgebra de Boole binaria, es el álgebra de Boole de la lógica booleana.

En muchos textos se identifica el concepto más abstracto de álgebra de Boole con esta álgebra en particular, lo cual no es correcto, ya que el concepto de álgebra de Boole es más abstracto y se aplica a estructuras diferentes de la lógica booleana, aunque compartan las mismas propiedades (no comparten los mismos elementos y operaciones). 

2) En teoría de conjuntos

Si U es un conjunto no vacío, que se llama universo o dominio. El conjunto de todos los conjuntos que se pueden definir con elementos de U , llamémosle \mathcal{C} , con las operaciones binarias \cap (*intersección*) y \cup (*unión*), la operación unaria *complementario de un conjunto*, e interpretando 0 como el conjunto vacío \emptyset y 1 como el conjunto U , forma un álgebra de Boole.

$(\mathcal{C}, \cap, \cup, \overline{}, \emptyset, U)$ es el álgebra de Boole de los conjuntos de U . Esta álgebra de Boole tiene más de dos elementos si \mathcal{C} los tiene. Más adelante se estudia en detalle esta álgebra de Boole.

 La teoría de conjuntos se presenta en el módulo "Teoría de conjuntos básica" de esta asignatura.

3) En matemáticas y física

Muchos otros ejemplos en el ámbito de las matemáticas y la física que quedan fuera del objetivo de este capítulo.

Resumen

En este módulo didáctico hemos argumentado la necesidad de un lenguaje y una semántica formal (una lógica) para poder expresar de una manera libre de ambigüedades los conceptos en informática y en las ingenierías, en general; y hemos presentado la más simple de las lógicas con aplicación a la informática: la lógica booleana.

El lenguaje formal de la lógica booleana consiste en nombrar con símbolos cualquier expresión y en formalizar rigurosamente la manera como se generan nuevas expresiones a partir de las dadas con los operadores booleanos. Paralelamente a la presentación de los diferentes operadores, se han introducido los dos valores de significado booleanos, simbolizados con 0 y 1, es decir, se ha introducido la semántica booleana. Éste es el rasgo más característico de la lógica booleana: sólo tiene dos valores de significado o valores de verdad. Para generar los valores de verdad correspondientes a las expresiones complejas se usan las tablas de verdad; así, se han definido las de los operadores básicos y se ha explicado cómo construir las tablas de verdad de expresiones complejas.

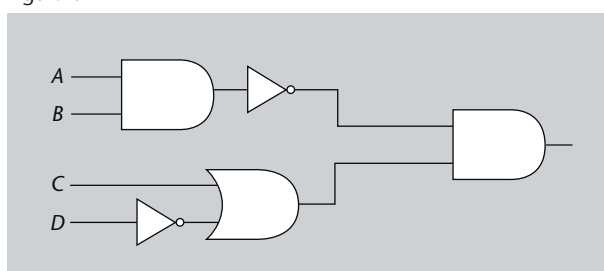
Posteriormente se han presentado algunas aplicaciones de la lógica booleana a la informática y se ha desarrollado la de los circuitos lógicos.

Finalmente, se ha definido la estructura matemática de álgebra de Boole como una generalización de la estructura que subyace a la lógica booleana. Un álgebra de Boole es un conjunto que contiene un 0 y un 1, y en el que se definen tres operaciones que cumplen las mismas propiedades que los operadores booleanos de negación, disyunción y conjunción. Así definida, la lógica booleana se puede ver como un álgebra de Boole con sólo dos elementos: 0 y 1.

Ejercicios de autoevaluación

- Demuestra que $(A \oplus B) \oplus B = A$, y que $(A \longleftrightarrow B) \longleftrightarrow B = A$; es decir, que $(A \oplus B) \oplus B$ y $(A \longleftrightarrow B) \longleftrightarrow B$ toman los mismos valores que A (independientemente de los valores de B).
- Evalúa el valor de la expresión $(A + B) \cdot (C + (\sim D))$ en los casos siguientes:
 - $A = 1, B = 1, C = 1, D = 1$
 - $A = 0, B = 0, C = 1, D = 1$
 - $A = 1, B = 1, C = 0, D = 0$
- Construyendo las correpondientes tablas de valor o tablas de verdad, prueba que en el caso de los operadores unarios de la lógica booleana, el operador negación junto a uno de los operadores constantes (por ejemplo el constante 1) permiten definir todos los operadores unarios.
- Analiza el circuito lógico de la figura 6 y expresa algebraicamente en forma de suma y productos la función lógica que implementa.

Figura 6



- Representa el circuito lógico correspondiente a la expresión: $\sim A + ((B + C) \cdot (\sim D))$.
- Demuestra que se cumplen las siguientes propiedades de álgebra de Boole en el caso de la lógica booleana:
 - Leyes de De Morgan.
 - Idempotencia.
 - Complementación.
 - Absorción.
- Representa con puertas *NAND* y *NOR* las siguientes expresiones:
 - $\sim((A + B) \cdot C)$
 - $\sim(A \cdot B) + (A \cdot C)$
- Demuestra que \neg y \vee definen los operadores binarios \wedge , \uparrow , \downarrow y \oplus .
- Expresa en función de los operadores \sim y $+$:
 - $(A \cdot B) \supset ((\sim A) \cdot C)$
 - $A \cdot (B \cdot (C \cdot D))$
- Expresa en función de los operadores \neg y \wedge :
 - $(A \longrightarrow B) \vee (\neg C)$
 - $(\neg A) \longrightarrow ((B \vee C) \longrightarrow (\neg D))$

Solucionario

1.

A	B	$A \oplus B$	$(A \oplus B) \oplus B$		A	B	$A \longleftrightarrow B$	$(A \longleftrightarrow B) \longleftrightarrow B$
1	1	0	1		1	1	1	1
1	0	1	1	y	1	0	0	1
0	1	1	0		0	1	0	0
0	0	0	0		0	0	1	0

2. La solución es:

- a) $(A + B) \cdot (C + (\sim D)) = 1$
 b) $(A + B) \cdot (C + (\sim D)) = 0$
 c) $(A + B) \cdot (C + (\sim D)) = 1$.

Se calcula con la tabla de valor siguiente correspondiente a los tres casos:

	A	B	C	D	$A + B$	$C + (\sim D)$	$(A + B) \cdot (C + (\sim D))$
a)	1	1	1	1	1	1	1
b)	0	0	1	1	0	1	0
c)	1	1	0	0	1	1	1

3. Recordad que los cuatro operadores unitarios posibles son los siguientes:

A	C_1	Id	\sim	C_0
1	1	1	0	0
0	1	0	1	0

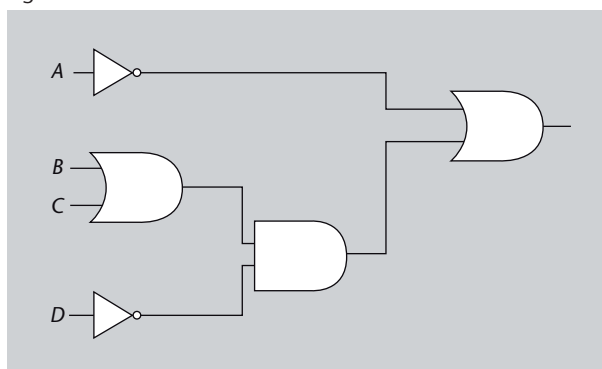
Obsérvese que Id es la negación de \sim , y que C_0 es la negación de C_1 , esto es:
 $IdA = \sim\sim A$ y $C_0A = \sim C_1A$.

Por tanto, los operadores \sim y C_1 definen todos los operadores unarios.

4. $\sim(A \cdot B) \cdot (C + (\sim D))$

5.

Figura 7



6.

a) Leyes de De Morgan

$$\sim(X + Y) = (\sim X) \cdot (\sim Y)$$

$$\sim(X \cdot Y) = (\sim X) + (\sim Y)$$

X	Y	$\sim(X + Y)$	$(\sim X) \cdot (\sim Y)$		X	Y	$\sim(X \cdot Y)$	$(\sim X) + (\sim Y)$
1	1	0	0	y	1	1	1	1
1	0	0	0		1	0	0	0
0	1	0	0		0	1	0	0
0	0	1	1		0	0	0	0

b) Idempotencia

$$X + X = X$$

$$X \cdot X = X$$

X	$X + X$	$X \cdot X$
1	1	1
0	0	0

c) Complementación. Se dice que $\sim x$ es el complementario de x

$$X + (\sim X) = 1$$

$$X \cdot (\sim X) = 0$$

X	$X + (\sim X)$	$X \cdot (\sim X)$
1	1	0
0	1	0

d) Absorción

$$X + (X \cdot Y) = X$$

$$X \cdot (X + Y) = X$$

X	Y	$X + (X \cdot Y)$	$X \cdot (X + Y)$
1	1	1	1
1	0	1	1
0	1	0	0
0	0	0	0

7.

a) $\sim((A + B) \cdot C) = \sim(A + B) + (\sim C)$ por una de las leyes de De Morgan de las álgebras de Boole, y ahora lo representamos con NOR y NAND (figura 8).

b) Lo representamos solo con NAND (figura 9).

8.

a) Sabemos ya que el operador conjunción se expresa en función de la negación y la disyunción:

$$A \wedge B = \neg((\neg A) \vee (\neg B))$$

Figura 8

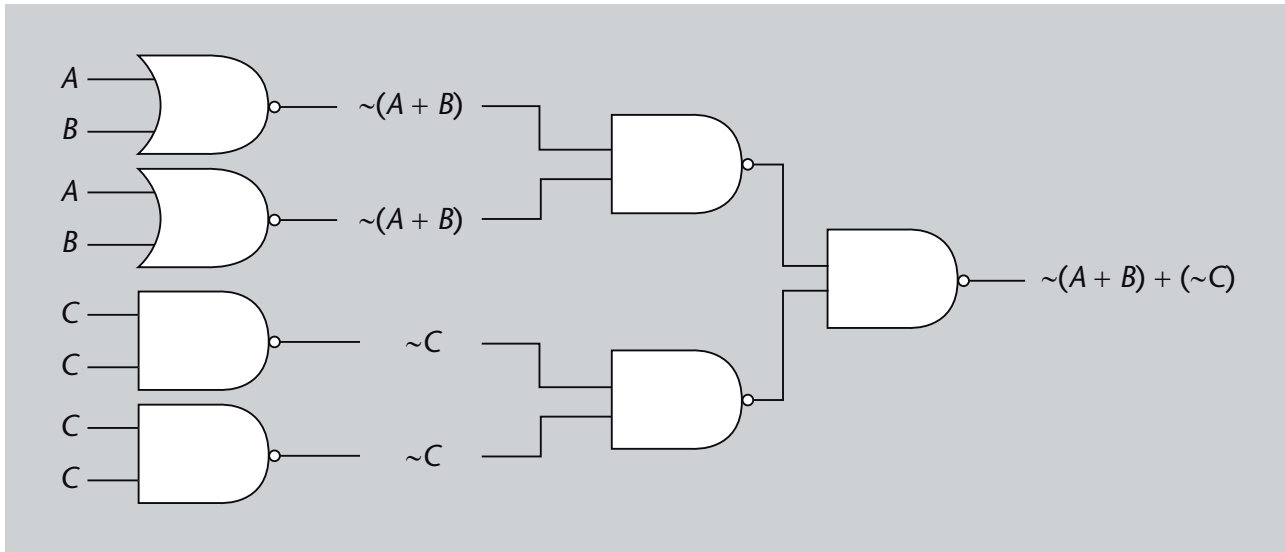
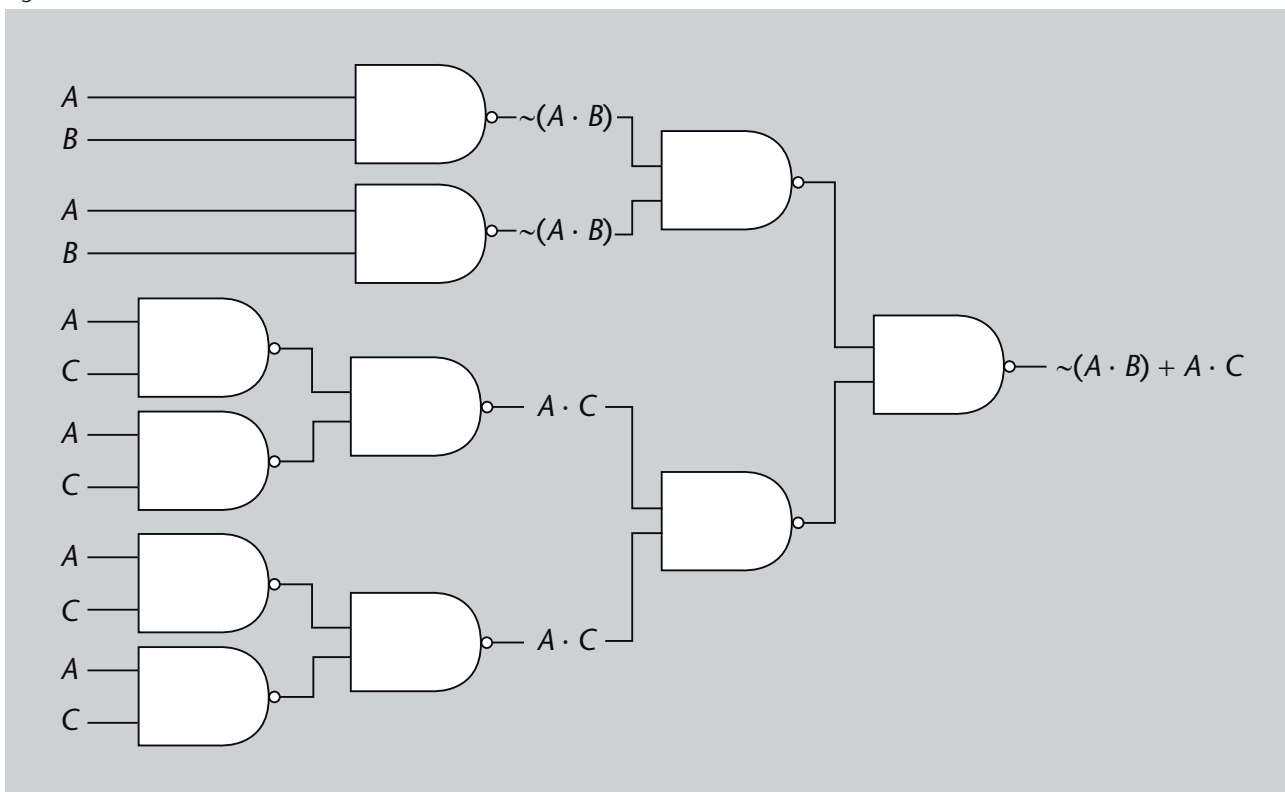


Figura 9



b) Sabemos que el operador *nand* es la negación del operador conjunción

$$A \uparrow B = \neg(A \wedge B)$$

Sustituyendo en el anterior la expresión de \wedge en función de \neg y \vee tenemos

$$\neg(A \wedge B) = \neg(\neg(\neg A) \vee (\neg B))$$

Aplicando la propiedad de la doble negación tenemos que

$$\neg\neg((\neg A) \vee (\neg B)) = (\neg A) \vee (\neg B).$$

Por tanto, se ha obtenido que

$$A \uparrow B = (\neg A) \vee (\neg B)$$

c) Sabemos que el operador *nor* es la negación del operador disyunción y, por tanto, se expresa así en función de la negación y la disyunción:

$$A \downarrow B = \neg(A \vee B)$$

d) Sabemos que el operador \oplus es la negación del operador \longleftrightarrow y, por tanto,

$$A \oplus B = \neg(A \longleftrightarrow B).$$

También sabemos que \longleftrightarrow se expresa en función de \longrightarrow y de \wedge de la forma

$$A \longleftrightarrow B = (A \longrightarrow B) \wedge (B \longrightarrow A) \text{ y, por tanto,}$$

$$\neg(A \longleftrightarrow B) = \neg((A \longrightarrow B) \wedge (B \longrightarrow A))$$

Ahora aplicamos la ley de De Morgan correspondiente, que permite expresar \neg y \wedge en función de \neg y \vee : $(\sim(x \cdot y) = (\sim x) \cdot (\sim y))$ y tenemos que

$$\neg((A \longrightarrow B) \wedge (B \longrightarrow A)) = (\neg(A \longrightarrow B)) \vee (\neg(B \longrightarrow A))$$

Sabemos que \longrightarrow se expresa en función de \neg y \vee ($A \longrightarrow B = \neg A \vee B$) y, por tanto:

$$(\neg(A \longrightarrow B)) \vee (\neg(B \longrightarrow A)) = (\neg(\neg A \vee B)) \vee (\neg(\neg B \vee A))$$

Aplicamos la ley de De Morgan de esta manera: $\neg(\neg A \vee B) = \neg\neg A \wedge \neg B$ y $\neg(\neg B \vee A) = \neg\neg B \wedge \neg A$;

que eliminando las dobles negaciones queda $\neg(\neg A \vee B) = A \wedge \neg B$ y $\neg(\neg B \vee A) = B \wedge \neg A$.

Sustituyendo en la expresión que estamos estudiando tenemos:

$$(\neg(\neg A \vee B)) \vee (\neg(\neg B \vee A)) = (A \wedge \neg B) \vee (B \wedge \neg A)$$

Veréis que ha vuelto a aparecer la conjunción, que de nuevo debemos expresar en función de \neg y \vee y tenemos:

$$(A \wedge \neg B) \vee (B \wedge \neg A) = \neg((\neg A) \vee (\neg\neg B)) \vee \neg((\neg B) \vee (\neg\neg A))$$

Finalmente, aplicando la propiedad de la doble negación, eliminamos las dos negaciones seguidas y queda:

$$A \oplus B = \neg((\neg A) \vee B) \vee \neg(\neg B \vee A)$$

Para comprobar que el resultado es correcto, basta construir la tabla de verdad de las dos expresiones y comprobar que el resultado de las respectivas columnas es el mismo.

A	B	$A \oplus B$	$(\neg A) \vee B$	$\neg((\neg A) \vee B)$	$(\neg B) \vee A$	$\neg((\neg B) \vee A)$	$\neg((\neg A) \vee B) \vee \neg((\neg B) \vee A)$
1	1	0	1	0	1	0	0
1	0	1	0	1	1	0	1
0	1	1	1	0	0	1	1
0	0	0	1	0	1	0	0

9.

$$\mathbf{a)} (A \cdot B) \supset ((\sim A) \cdot C) =$$

$$= \sim(A \cdot B) + ((\sim A) \cdot C) =$$

$$= ((\sim A) + (\sim B)) + ((\sim A) \cdot C) =$$

$$= (\sim A) + (\sim B) + (\sim((\sim\sim A) + (\sim C))) =$$

$$= ((\sim A) + (\sim B)) + (\sim(A + (\sim C)))$$

Comprobación:

A	B	C	$(A \cdot B) \supset ((\sim A) \cdot C)$	$((\sim A) + (\sim B)) + (\sim(A + (\sim C)))$
1	1	1	0	0
1	1	0	0	0
1	0	1	1	1
1	0	0	1	1
0	1	1	1	1
0	1	0	1	1
0	0	1	1	1
0	0	0	1	1

$$\mathbf{b)} A \cdot (B \cdot (C \cdot D)) =$$

$$= \sim((\sim A) + (\sim(B \cdot (C \cdot D)))) =$$

$$= \sim((\sim A) + ((\sim B) + (\sim(C \cdot D)))) =$$

$$= \sim((\sim A) + ((\sim B) + ((\sim C) + (\sim D))))$$

Comprobación:

A	B	C	D	$A \cdot (B \cdot (C \cdot D))$	$\sim((\sim A) + ((\sim B) + ((\sim C) + (\sim D))))$
1	1	1	1	1	1
1	1	1	0	0	0
1	1	0	1	0	0
1	1	0	0	0	0
1	0	1	1	0	0
1	0	1	0	0	0
1	0	0	1	0	0
1	0	0	0	0	0
0	1	1	1	0	0
0	1	1	0	0	0
0	1	0	1	0	0
0	1	0	0	0	0
0	0	1	1	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	0	0

10.

a) Como $A \rightarrow B = \neg(A \wedge (\neg B))$ tenemos que

$$(A \rightarrow B) \vee (\neg C) = ((\neg A) \wedge (\neg B)) \vee (\neg C).$$

Teniendo en cuenta la expresión de \vee en función de \wedge y \sim ($A \vee B = \neg((\neg A) \wedge (\neg B))$) tenemos que

$$(\neg(A \wedge (\neg B))) \vee (\neg C) = \neg(\neg\neg(A \wedge (\neg B)) \wedge (\neg\neg C)).$$

Eliminando las dobles negaciones tenemos que

$$\neg(\neg\neg(A \wedge (\neg B)) \wedge (\neg\neg C)) = \neg((A \wedge (\neg B)) \wedge C)$$

Comprobación

A	B	C	$(A \rightarrow B) \vee (\neg C)$	$\neg((A \wedge (\neg B)) \wedge C)$
1	1	1	1	1
1	1	0	1	1
1	0	1	0	0
1	0	0	1	1
0	1	1	1	1
0	1	0	1	1
0	0	1	1	1
0	0	0	1	1

$$\begin{aligned}
 \text{b) } & (\neg A) \rightarrow ((B \vee C) \rightarrow (\neg D)) = \\
 & = \neg(\neg A) \vee ((B \vee C) \rightarrow (\neg D)) = \\
 & = A \vee ((B \vee C) \rightarrow (\neg D)) = \\
 & = \neg(\neg A \wedge \neg((B \vee C) \rightarrow (\neg D))) = \\
 & = \neg(\neg A \wedge \neg(\neg(B \vee C) \vee (\neg D))) = \\
 & = \neg(\neg A \wedge \neg(\neg B \wedge \neg C) \vee (\neg D)) = \\
 & = \neg(\neg A \wedge \neg(\neg B \wedge \neg C) \wedge (\neg\neg D)) = \\
 & = \neg(\neg A \wedge \neg(\neg B \wedge \neg C) \wedge D)
 \end{aligned}$$

Comprobación

A	B	C	D	$(\neg A) \longrightarrow ((B \vee C) \longrightarrow (\neg D))$	$\neg (\neg A \wedge \neg (\neg B \wedge \neg C) \wedge D))$
1	1	1	1	1	1
1	1	1	0	1	1
1	1	0	1	1	1
1	1	0	0	1	1
1	0	1	1	1	1
1	0	1	0	1	1
1	0	0	1	1	1
1	0	0	0	1	1
0	1	1	1	0	0
0	1	1	0	1	1
0	1	0	1	0	0
0	1	0	0	1	1
0	0	1	1	0	0
0	0	1	0	1	1
0	0	0	1	1	1
0	0	0	0	1	1

Glosario

conjunción

Operador booleano binario.

disyunción

Operador booleano binario.

función lógica

Nombre de los operadores booleanos en el contexto de la computación.

implicación

Operador booleano binario.

nand

Operador booleano binario.

negación

Operador booleano unario.

nor

Operador booleano binario.

operador binario

El que opera sobre dos expresiones.

operador booleano

El que opera con dos valores de verdad.

operadores primitivos

Conjunto de operadores que pueden definir todos los demás.

puerta lógica

En un circuito lógico, es la representación de la conversión de la señal de entrada en la señal de salida.

tabla de verdad

Algoritmo de cálculo del valor de verdad de una expresión compleja a partir de las que la componen.

valor de significado (de verdad)

Valor asignado a las expresiones en una lógica.

Bibliografía

Ben-Ari, M. (2004). *Mathematical Logic for Computer Science*. Springer.

Gajsky, D.D. (1997). *Principios de Diseño Digital*. Prentice Hall.

Martín, F., Sánchez, J.L., Paniagua, E. (2003). *Lógica computacional*. Paraninfo.

Roth, C.H. (2005). *Fundamentos de diseño lógico*. Thomson.