**UNIVERSIDAD NACIONAL SAN AGUSTÍN DE AREQUIPA**

**FACULTAD DE INGENIERÍA DE PROCESOS Y SERVICIOS**

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS



**“ESTRUCTURAS DISCRETAS 1”**

**INFORME: ÁLGEBRA DE BOOLE**

Docente:

**ING. OLHA SHARHORODSKA**

* Chahuara Mamani Marco Steven CUI: 20240688
* Escalante Quispe Yhon Deivis CUI: 20240695
* Subia Huaicane Edson Fabricio CUI: 20240679
* Llaique Chullunquia Jack Franco CUI: 20243399

Arequipa – 2024

### **Índice**

### **Introducción**

### **Historia del Álgebra de Boole**

### **Conceptos Básicos**

### **Operaciones Fundamentales**

### **Propiedades del Álgebra de Boole**

### **Teoremas Importantes**

### **Aplicaciones del Álgebra de Boole**

### **Ejemplos Prácticos**

### **Implementaciones en Circuitos Lógicos.**

### **Simplificación de Expresiones Booleanas**

### **Métodos de Simplificación: Mapas de Karnaugh**

### **Álgebra de Boole y Computación**

### **Conclusiones**

### **Referencias**

### 

### 

### 

### 

### **1. Introducción**

El Álgebra de Boole es una rama fundamental de las matemáticas que se utiliza para describir y manipular relaciones lógicas. Es esencial en la electrónica y la computación, especialmente en el diseño y análisis de circuitos digitales. En el contexto de la Ingeniería de Sistemas, su importancia no puede ser subestimada, ya que proporciona las herramientas necesarias para simplificar y resolver problemas complejos relacionados con los sistemas digitales.

Los circuitos que forman una computadora son variados: algunos están destinados a proporcionar la energía necesaria para las diferentes partes de la máquina, mientras que otros se encargan de generar, procesar y transmitir señales que contienen información. Dentro de este segundo grupo, se pueden diferenciar los circuitos que manejan información analógica y aquellos que trabajan con valores digitales, utilizando el Álgebra de Boole.

El Álgebra de Boole se basa en la lógica y se utiliza para representar circuitos lógicos en forma de ecuaciones. Es una herramienta poderosa que soluciona y simplifica todos los problemas relacionados con los sistemas digitales. Esta es particularmente importante para la programación, ya que permite desarrollar algoritmos y estructuras de control condicional. Con el Álgebra de Boole, se pueden revisar procesos, tomar decisiones y gestionar el flujo de proyectos de manera eficiente.

A diferencia del álgebra tradicional, que se basa en números y operaciones aritméticas, el Álgebra de Boole utiliza variables booleanas (0 y 1) y operadores lógicos (AND, OR, NOT) para representar y manipular datos. Por lo tanto, también se conoce como "álgebra de conmutación". Podemos representar el funcionamiento de los circuitos lógicos utilizando números, siguiendo algunas reglas conocidas como "Leyes del Álgebra de Boole".

Es crucial recordar que el Álgebra de Boole es muy diferente al álgebra matemática regular y sus métodos. Comprender estas diferencias y aplicar correctamente las leyes del Álgebra de Boole es esencial para cualquier estudiante de Ingeniería de Sistemas, ya que facilita el diseño y la optimización de sistemas tanto de hardware como de software. Este informe explorará los conceptos básicos, operaciones fundamentales, propiedades, teoremas importantes y aplicaciones del Álgebra de Boole, destacando su relevancia en el campo de la Ingeniería de Sistemas.

### 

### **2. Historia del Álgebra de Boole (1 página)**

El álgebra de Boole fue desarrollada por George Boole, un matemático y lógico británico, en el siglo XIX. En 1854, Boole publicó su obra principal, "An Investigation of the Laws of Thought", donde introdujo un sistema algebraico diseñado para resolver problemas de lógica. Este sistema se basaba en operaciones simples aplicables a valores de verdad, sentando las bases de lo que hoy conocemos como Álgebra de Boole.

Inicialmente, el trabajo de Boole no tuvo una gran influencia, pero con el tiempo se convirtió en la piedra angular del diseño de circuitos digitales y la teoría de la computación, campos esenciales en la Ingeniería de Sistemas. Boole fue pionero al demostrar cómo los principios matemáticos podían aplicarse a la lógica y al razonamiento, estableciendo las bases para la informática moderna.

Después de la publicación de su libro, el Álgebra de Boole empezó a ser reconocida como la forma perfecta para representar los circuitos lógicos digitales. A fines del siglo XIX, científicos como William Stanley Jevons, Ernst Schröder y Edward Vermilye Huntington contribuyeron a modernizar y expandir el concepto. En 1936, Marshall Harvey Stone demostró que el Álgebra de Boole es 'isomorfa' a los conjuntos, una importante área de las matemáticas.

En la década de 1930, Claude Shannon, un científico estadounidense, desarrolló un nuevo método conocido como "Cambio de Álgebra" utilizando los conceptos del Álgebra de Boole para estudiar los circuitos de conmutación. Shannon demostró cómo las operaciones booleanas podían aplicarse al diseño y análisis de sistemas de conmutación, marcando el comienzo de la teoría moderna de la información y los circuitos digitales.

La síntesis lógica en las herramientas modernas de automatización electrónica se representa de manera eficiente mediante el uso de funciones booleanas conocidas como "Diagramas de Decisión Binarios" (BDD). Estos diagramas permiten la representación y simplificación de expresiones booleanas complejas, facilitando el diseño de circuitos lógicos.

### **3. Conceptos Básicos (1.5 páginas)**

El Álgebra de Boole es una estructura matemática que se utiliza para analizar y diseñar sistemas digitales mediante el uso de variables booleanas y operaciones lógicas. En su esencia, el Álgebra de Boole se basa en el concepto de variables que solo pueden asumir dos valores posibles: verdadero (1) y falso (0). Estos valores se utilizan para formar expresiones booleanas mediante operaciones lógicas, que permiten representar y manipular información en sistemas digitales y computacionales.

**Variables Booleanas**

Las variables booleanas son los elementos fundamentales del Álgebra de Boole. Estas variables, como A, B y C, pueden tener solo dos valores: 0 (falso) y 1 (verdadero). Cada variable booleana representa un estado particular en un sistema lógico o digital. La capacidad de estas variables para tomar solo dos valores es crucial para el diseño de circuitos digitales y la programación de sistemas informáticos.

**Valores de Verdad**

Los valores de verdad son una forma de representar la información en el Álgebra de Boole. En este contexto, el valor 0 indica una condición falsa, mientras que el valor 1 indica una condición verdadera. Estos valores se utilizan en tablas de verdad, que son herramientas esenciales para evaluar y comprender el comportamiento de las expresiones booleanas y las operaciones lógicas. Las tablas de verdad ayudan a visualizar cómo diferentes combinaciones de entradas afectan las salidas en sistemas digitales.

**Relación con la Ingeniería de Sistemas**

En Ingeniería de Sistemas, los conceptos básicos del Álgebra de Boole son fundamentales para entender el funcionamiento de los sistemas digitales. Las variables booleanas representan los estados de los circuitos electrónicos, mientras que las operaciones lógicas determinan cómo estos estados interactúan para producir salidas deseadas. Por ejemplo, en el diseño de circuitos electrónicos, las variables booleanas se utilizan para definir el comportamiento de puertas lógicas como AND, OR y NOT, que son los bloques de construcción básicos de cualquier sistema digital.

Además, el Álgebra de Boole es crucial para la programación de sistemas, ya que los algoritmos y las estructuras de control en los lenguajes de programación a menudo se basan en operaciones booleanas. Las expresiones booleanas permiten a los programadores crear condiciones y realizar decisiones dentro del código, facilitando la implementación de lógica compleja en aplicaciones de software.

### **4. Operaciones Fundamentales (2 páginas)**

Las operaciones fundamentales del Álgebra de Boole son AND, OR y NOT. Cada una de estas operaciones tiene una tabla de verdad que define sus resultados para todas las combinaciones posibles de sus entradas.

* **Operación AND**: Representada como A \* B o A AND B, da como resultado verdadero solo si ambas variables son verdaderas.
  + Tabla de verdad:

| **A** | **B** | **A AND B** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

* **Operación OR**: Representada como A + B o A OR B, da como resultado verdadero si al menos una de las variables es verdadera.
  + Tabla de verdad:

| **A** | **B** | **A OR B** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

* **Operación NOT**: Representada como ¬A o NOT A, invierte el valor de la variable.
  + Tabla de verdad:

| **A** | **NOT A** |
| --- | --- |
| 0 | 1 |
| 1 | 0 |

Estas operaciones son cruciales en el diseño de circuitos lógicos utilizados en computadoras y otros dispositivos electrónicos, ya que permiten la creación de sistemas que pueden tomar decisiones basadas en combinaciones de condiciones.

### **5. Propiedades del Álgebra de Boole (1.5 páginas)**

El Álgebra de Boole posee varias propiedades que facilitan la simplificación y manipulación de expresiones booleanas. Las principales propiedades son:

* **Propiedad Conmutativa**:
  + AND: A \* B = B \* A
  + OR: A + B = B + A
* **Propiedad Asociativa**:
  + AND: (A \* B) \* C = A \* (B \* C)
  + OR: (A + B) + C = A + (B + C)
* **Propiedad Distributiva**:
  + AND sobre OR: A \* (B + C) = (A \* B) + (A \* C)
  + OR sobre AND: A + (B \* C) = (A + B) \* (A + C)
* **Ley de Identidad**:
  + AND: A \* 1 = A
  + OR: A + 0 = A
* **Ley de Complementación**:
  + AND: A \* ¬A = 0
  + OR: A + ¬A = 1

Estas propiedades son herramientas esenciales para los ingenieros de sistemas, ya que permiten la simplificación de expresiones lógicas complejas, haciendo posible el diseño de circuitos más eficientes y la optimización de algoritmos.

### 

### 

### 

### **6. Teoremas Importantes**

* **Teorema 1**

Este teorema establece que para cualquier conjunto **A** en relación con un conjunto universal **U**, el complemento de **A**, denotado como **A′ o Aˉ**, es único. Esto significa que **A′** contiene exactamente los elementos de **U** que no están en **A**.Formalmente, **A′={x∈U∣x∉A}**.

* **Teorema 2 (ELEMENTOS NULOS)**

Es una afirmación fundamental en el álgebra de Boole que establece reglas sobre cómo interactúan las operaciones de unión (+) e intersección (⋅) con los elementos identidad (1) y nulos (0).

* El teorema **A+1=1** muestra que la unión de cualquier conjunto con el conjunto universal 1 no cambia el conjunto original.
* El teorema **A⋅0=0** muestra que la intersección de cualquier conjunto con el conjunto vacío 0 resulta en el conjunto vacío 0.
* **Teorema 3**
* **0′=1**

Aquí, 0′ representa el complemento del conjunto vacío 0. El complemento del conjunto vacío es el conjunto universal 1. Esto significa que el conjunto que contiene todos los elementos del universo **U** es el complemento del conjunto vacío, ya que no hay elementos en el conjunto vacío y el conjunto universal incluye todos los elementos posibles.

* **1′=0**

En este caso, 1′ representa el complemento del conjunto universal 1. El complemento del conjunto universal es el conjunto vacío 0, ya que el conjunto universal incluye todos los elementos posibles, y su complemento no incluye ninguno.

* **Teorema 4 (IDEMPOTENCIA)**

Este teorema establece dos propiedades fundamentales de los conjuntos respecto a las operaciones de unión (+) e intersección (⋅):

* **A+A=A**

Esta propiedad indica que la unión de un conjunto **A** consigo mismo es igual a **A** mismo. Es decir, la operación de unión (+) de un conjunto con él mismo no cambia el conjunto original. Esto es intuitivo porque unir un conjunto consigo mismo simplemente resulta en el mismo conjunto sin duplicaciones adicionales.

* **A⋅A=A**

Esta propiedad establece que la intersección de un conjunto **A** consigo mismo es igual a **A** mismo. Es decir, la operación de intersección (⋅) de un conjunto con él mismo no cambia el conjunto original. Esto se debe a que la intersección solo incluye los elementos que están presentes en ambos conjuntos, y si un conjunto está presente en sí mismo, entonces todos sus elementos están también en el conjunto intersecado.

* **Teorema 5 (INVOLUCIÓN)**

Este teorema establece una propiedad importante de los conjuntos en relación con su complemento

* **(A’)’ = A**

Esto significa que el complemento del complemento de un conjunto **A** es igual al conjunto **A** mismo. En otras palabras, si tomamos el complemento de un conjunto **A**, y luego tomamos el complemento de ese resultado nuevamente, obtendremos el conjunto **A** original.

* **Teorema 6 (ABSORCIÓN)**

Este teorema establece dos propiedades importantes respecto a las operaciones de unión (+) e intersección (⋅):

* **A+A·B=A**

Esta propiedad indica que la unión de un conjunto **A** con la intersección de **A** y **B** es igual a **A** mismo. Es decir, si **A** está incluido en **B**, entonces unir **A** con la intersección de **A** y **B** no cambia el conjunto original **A**. Esto se debe a que la intersección **A⋅B** solo añade elementos que están en ambos conjuntos **A** y **B**, por lo que la unión con **A** no agrega elementos adicionales a **A**.

* **A·(A+B)=A**

Esta propiedad establece que la intersección de **A** con la unión de **A** y **B** es igual a **A** mismo. Es decir, si **A** está incluido en **B**, entonces intersectar **A** con la unión de **A** y **B** no cambia el conjunto original **A**. Esto se debe a que la unión **A+B** incluye todos los elementos que están en **A** o en **B**, y la intersección **A⋅(A+B)** solo conserva los elementos que están en ambos conjuntos **A** y **A+B**, lo cual es equivalente a solo **A**.

* **Teorema 7**

Este teorema establece dos propiedades importantes respecto a las operaciones de unión (+) e intersección (⋅):

* **A + A’·B = A + B**

Esta propiedad indica que la unión de **A** con el producto de su complemento **A′** y **B** es igual a la unión de **A** con **B**. En otras palabras, agregar **A′⋅B** a **A** no cambia el resultado de unir **A** con **B**. Esto se debe a que si **B** contieneelementos que no están en **A, A′⋅B** solo añadirá estos elementos a **A**, pero la unión con **A** incluirá automáticamente todos los elementos de **B,** por lo que no hay necesidad de incluir **A′⋅B** adicionalmente.

* **A · (A’ + B) = A · B**

Esta propiedad establece que la intersección de **A** con la suma de su complemento **A′**y **B** es igual a la intersección de **A** con **B**. En otras palabras, intersectar **A** con **A′+B** es lo mismo que intersectar **A** con **B**. Esto se debe a que **A′+B** representa todos los elementos que están en **B** pero no en **A**, y la intersección con **A** solo mantendrá los elementos que están en ambos conjuntos **A** y **B**.

* **Teorema 8 (ASOCIATIVIDAD)**

Cada uno de los operadores binarios (+) y (·) cumple la propiedad asociativa:

* **A+(B+C) = (A+B)+C**

Esta propiedad indica que la asociación de la unión de tres conjuntos **A**, **B**, y **C** no depende del orden en que se realicen las operaciones. En otras palabras, primero podemos realizar la unión de **B** y **C**, y luego unir el resultado con **A**, o podemos unir **A** y **B** primero, y luego unir el resultado con **C**, obteniendo el mismo conjunto final en ambos casos.

* **A·(B·C) = (A·B)·C**

Esta propiedad establece que la asociación de la intersección de tres conjuntos **A**, **B**, y **C** tampoco depende del orden en que se realicen las operaciones. En otras palabras, primero podemos realizar la intersección de **B** y **C**, y luego intersectar el resultado con **A**, o podemos intersectar **A** y **B** primero, y luego intersectar el resultado con **C**, obteniendo el mismo conjunto final en ambos casos.

* **Leyes de Morgan**

Las leyes de De Morgan en el álgebra de Boole son dos principios fundamentales que describen la relación entre el complemento de una operación de unión (+++) o intersección (⋅\cdot⋅) y las operaciones complementarias de los conjuntos individuales:

* **(A+B)’ = A’·B’**

Esta ley establece que el complemento de la unión de dos conjuntos **A** y **B** es igual a la intersección de los complementos de **A** y **B**. En otras palabras, si un elemento no está en **A** ni en **B**, entonces no está en la unión de **A** y **B**.

* **(A·B)’ = A’ + B’**

Esta ley establece que el complemento de la intersección de dos conjuntos **A** y **B** es igual a la unión de los complementos de **A** y **B**. En otras palabras, si un elemento no está en **A** y tampoco está en **B**, entonces no está en la intersección de **A** y **B**.

### **7. Aplicaciones del Álgebra de Boole**

* **Puertas lógicas:**

Las puertas lógicas son circuitos electrónicos que realizan operaciones lógicas en una o más señales de entrada para producir una señal de salida.La señal de salida depende de la función lógica realizada por la puerta, que está determinada por su expresión booleana.Por ejemplo, AN And Gate produce una señal de salida solo cuando ambas entradas son altas.La expresión booleana para una puerta y es A.B, donde A y B son las señales de entrada.

* **Circuitos digitales:**

Los circuitos digitales son los bloques de construcción de sistemas digitales modernos, como computadoras, teléfonos inteligentes y otros dispositivos electrónicos.Están compuestos por puertas lógicas que están conectadas para realizar una función específica.El uso de álgebra booleana en circuitos digitales permite a los diseñadores optimizar el rendimiento del circuito simplificando las expresiones lógicas y minimizando el número de puertas necesarias para implementar el circuito.

* **Microprocesadores:**

Los microprocesadores son el corazón de todas las computadoras modernas.Son circuitos digitales complejos que realizan una variedad de funciones, como operaciones aritméticas, lógicas y de control.El uso de álgebra booleana en microprocesadores permite a los diseñadores optimizar el rendimiento del procesador al reducir el número de puertas lógicas y minimizar el consumo de energía.

* **Redes informáticas:**

Las redes informáticas están compuestas por una gran cantidad de dispositivos digitales interconectados, como computadoras, servidores y enrutadores.El uso de álgebra booleana en redes informáticas permite a los diseñadores optimizar el rendimiento de la red simplificando las expresiones lógicas y minimizando el número de puertas necesarias para implementar la red.

* **Diseño de circuitos digitales**:

Desempeñando un papel crucial en la creación y optimización de procesadores, memorias y dispositivos lógicos programables. Permite la representación y manipulación formal de la lógica booleana, facilitando la construcción de circuitos que realizan operaciones complejas mediante la combinación de puertas lógicas básicas como AND, OR y NOT. Esta metodología es esencial para garantizar la fiabilidad, eficiencia y funcionamiento preciso de los sistemas digitales modernos, desde microprocesadores hasta sistemas embebidos y redes de comunicación.

* **Programación y algoritmos**:

Facilita la capacidad de tomar decisiones y controlar el flujo de ejecución dentro de los programas mediante la lógica booleana. Esto se logra mediante la utilización de expresiones booleanas que permiten evaluar condiciones y ejecutar acciones basadas en resultados verdaderos o falsos. Así, los algoritmos pueden implementar estructuras de control como bucles y condicionales de manera eficiente y precisa, asegurando que los programas funcionen según lo previsto y respondan adecuadamente a diferentes situaciones y entradas de datos.

* **Sistemas de control y automatización**:

Especialmente en aplicaciones industriales como la industria manufacturera. Permite representar de manera formal las relaciones lógicas entre variables de entrada y salida en sistemas de automatización y control. Esto facilita la creación de modelos matemáticos precisos que describen el comportamiento de los sistemas bajo diferentes condiciones operativas.

En el contexto de sistemas de control, el álgebra de Boole se utiliza para definir las funciones lógicas que determinan cómo deben manipularse las señales de entrada para lograr los resultados deseados en las salidas. Estas funciones son fundamentales para la implementación de lógica de control en PLCs (Controladores Lógicos Programables) y otros dispositivos de automatización industrial. Además, el álgebra de Boole permite la simplificación y optimización de circuitos lógicos utilizados en los sistemas de control, lo cual es crucial para garantizar la eficiencia y fiabilidad del proceso de automatización en las líneas de producción y otros entornos industriales.

* **Criptografía**:

El álgebra de Boole permite definir operaciones lógicas que son esenciales para la implementación de algoritmos criptográficos, como las funciones de cifrado y descifrado. Estas funciones utilizan operaciones booleanas básicas para transformar los datos de manera que sean incomprensibles para cualquier persona que no posea la clave adecuada. Además, el álgebra de Boole se utiliza para la generación de claves criptográficas y para la verificación de la seguridad de los protocolos criptográficos.

El álgebra booleana es una herramienta esencial en el diseño, análisis y optimización de circuitos y sistemas digitales.Sus aplicaciones en computación y electrónica son diversas y de largo alcance, desde compuertas lógicas simples hasta microprocesadores complejos y redes informáticas.El uso de álgebra booleana ha revolucionado el campo de la Ingeniera en Sistemas y ha allanado el camino para el desarrollo de sistemas digitales modernos.

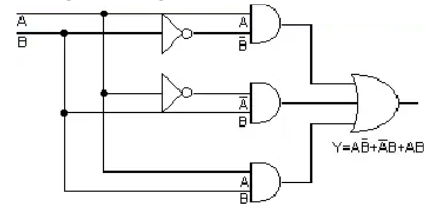
En resumen, el álgebra de Boole tiene una importante función en el mundo de la lógica y las ciencias de la computación. Sus leyes, ventajas y aplicaciones en el diseño de circuitos digitales, programación y otras áreas demuestran su relevancia y utilidad en el mundo moderno.

### **8.- Ejemplos Prácticos**

**Ejemplo 1:**

**Considerar la expresión booleana Y = A·B' + A'·B + A·B. Un diagrama lógico de ésta**

**expresión aparece en la siguiente figura:**



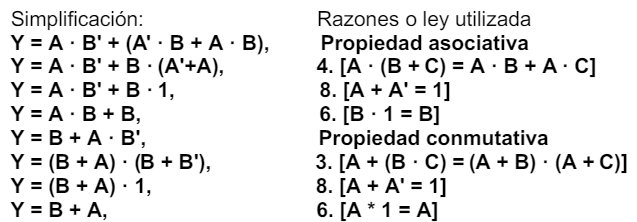
**Observar que deben utilizarse seis compuertas para implementar este circuito lógico,**

**que realiza la lógica detallada en la tabla de verdad que se muestra.**



Simplifiquemos auxiliándose con las leyes del Álgebra de Boole:

**Y = A·B' + A'·B + A·B**



Concluimos entonces que una sola puerta OR de dos entradas realiza la misma función

**Ejemplo 2:**

Pasar la expresión booleana en forma de maxitérminos:

**Y = (C + B + A) · (C + B' + A) · (C + B' + A') · (C' + B + A) · (C' + B + A') · (C' + B'+ A)**

a su correspondiente en forma de minitérminos:

**Y = C' · B' · A + C · B · A**

tenemos :

**Y = (C + B + A) · (C + B' + A) · (C + B' + A') · (C' + B + A) · (C' + B + A') · (C' + B' + A)**

**Y = [ (C + B + A) · (C + B' + A) ] · [ (C + B' + A') · (C' + B + A') ] · [ (C' + B + A) · (C' + B' +A) ], Aplicando Propiedad asociativa y conmutativa**

**Y = { [ (C + A) + B ] · [ (C + A) + B' ] } · { [ (C + B') + A' ] · [ (C' + B) + A' ] } · { [ (C' + A) +**

**B ] · [ (C' + A) + B' ] }, Aplicando Propiedad asociativa y conmutativa.**

**Y = [ (C + A) + B · B' ] · [ (C + B') · (C' + B) + A' ] · [ (C' + A) + B · B' ] - - - - , Aplicando:**

**A + (B · C) = (A + B) · (A + C).**

**Y = (C + A) · [ (C + B') · (C' + B) + A' ] · (C' + A) - - - , Aplicando [ A · A' = 0 ] y [ A + 0 =**

**A ].**

**Y = (C + A) · (C' + A) · [ (C + B') · (C' + B) + A' ] - - - , Aplicando Propiedad conmutativa.**

**Y = (C · C' + A) · [ (C + B') · C' + (C + B') · B + A' ] - - - , Aplicando: [ A + (B · C) = (A +**

**B)(A + C) ] y [ A · (B + C) = A · B + A · C ].**

**Y = A · [ C' · C + C' · B' + C · B + B · B' + A' ]- - - , Aplicando: [ A · A' = 0 ] , [ A · (B + C)**

**= A · B + A · C ] y [ A + 0 = A ].**

**Y = A · [ C' · B' + C · B + A' ] - - - - - - - - - - - , Aplicando: [ A · A' = 0 ] y [ A + 0 = A ].**

**Y = A · C' · B' + A · C · B + A · A' ] - - - - - , Aplicando: [ A · (B + C) = A · B + A · C ].**

**Y = A · C' · B' + A · C · B - - - - - - - - - , Aplicando: [ A · A' = 0 ] y [ A + 0 = A ].**

**Y = C' · B' · A + C · B · A - - - - - - - - , Aplicando Propiedad conmutativa.**

**Ejemplo 3:**

Reducir la siguiente expresión:

**F = A · B' + A · B · C + A · B' · C + A · B**

Sacando factor común

**A · B'**

a los términos primero y tercero y

**A · B**

a los términos segundo y cuarto:

**F = A · B' · (1 + C) + A · B · (C + 1)**

**F = A · B' · 1 + A · B · 1 = A · B' + A · B**

Sacando por último factor común A, se obtiene:

**F = A · (B' + B) = A · 1**

**F = A**

**Ejemplo 4:**

Aplicar las leyes de DeMorgan a la siguiente ecuación:

**F = A' B' C + A D + B' C D'**

Si sacamos factor común **B' C** en el primero y tercer términos:

**F = B' C (A' +D') + A D**

Observando la forma dual de la Ley de DeMorgan, nos damos cuenta que el contenido del paréntesis de la ecuación **(A' +D')** proviene de la aplicación de dicha ley a **(A·D)',** con lo que podemos escribir:

**(A' + D') = (A · D)'**

y por tanto:

**F = B' C (A · D)' + A D**

Si ahora llamamos

**X = A D**

, se tiene

**F = B' C X' + X**

, de donde:

**F = B´C + X**

y deshaciendo el cambio de variable:

**F = B' C + A D**

**Ejemplo 5:**

Simplificar la siguiente ecuación y obtener la tabla de verdad que corresponde a su

funcionamiento:

**F = a b c' + a' b' d + a b'**

Sacando factor común "a" al primer y tercer sumandos y aplicando algebra de boole:

**F = a (b c' + b') + a b' d**

**F= a (b' +c') + a' b' d**

Denominando

**x = b' + c' e y = b' d,** y a través de la ley de transposición, resulta:

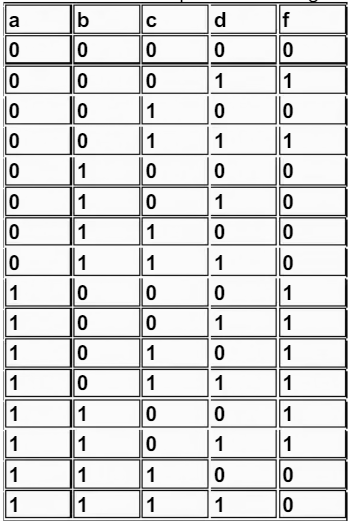
**F = a x + a' y = (a + y) · (a' + x)** y sustituyendo las **x** e **y:**

**F = (a + b' d) · (a'+b'+c')**

Por último, con la forma dual de la ley de DeMorgan en reversa,tenemos:

**F = (a + b' + d) · (a · b · c)'**

La tabla de verdad de la ecuación resultante queda como sigue:



**Ejemplo 6:**

Demostrar que

**(A + B) (A +C) = A +BC**

**(A + B) (A +C) = AA + AC + AB + BC,** Ley Distributiva,

**(A + B) (A +C) = A + AC + AB + BC,** por **AA = A,**

**(A + B) (A +C) = A(1 + C) + AB + BC,** sacando factor común **A,**

**(A + B) (A +C) = A + AB + BC,** aplicando **1 + C = 1 y A \*1 = A,**

**(A + B) (A +C) = A(1 + B) + BC,** sacando factor común **A,**

**(A + B) (A +C) = A + BC,** aplicando **1 + B = 1 y A \*1 = A.**

Y finalmente:

**(A + B) (A +C) = A + BC**

**Ejemplo 7:**

Simplificar

**x = abc + abcd + ac**

**x = abc + abcd + ac**

**x = abc (1 + d) + ac**

**x = abc + ac**

**x = ac (1 + b)**

**x = ac**

**Ejemplo 8:**

Simplificar

**x = acd + ac + abc**

**x = acd + ac + abc**

**x= ac + acd + abc + cd**

**x= ac +cd + abc**

**x= ac +cd + abc + ab**

**x = ac +cd + ab**

**Ejemplo 9:**

Simplificar:

**V = (W + WX’ + YZ)’**

**V = (W + WX’ + YZ)’**

**V = W’ (W’ + X) (Y’ + Z’)** Aplicando teorema de D’Morgan

**V = (W’W’ + W’X) (Y’ + Z’)** Realizando el producto de los dos primeros términos

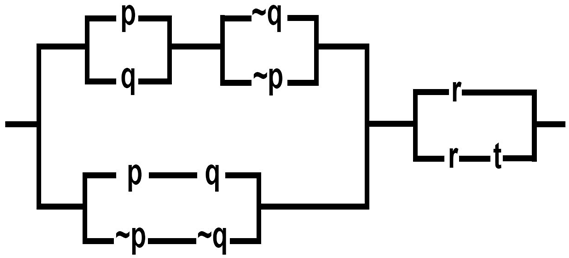
**V = (W’ + W’X) (Y’ + Z’)** simplificando variables

**V = (W’(1 + X)) (Y’ + Z’)** multiplicando los términos

**V = W’ (Y’ + Z’)** aplicando la regla de **X(1+Y) = X(1) = X**

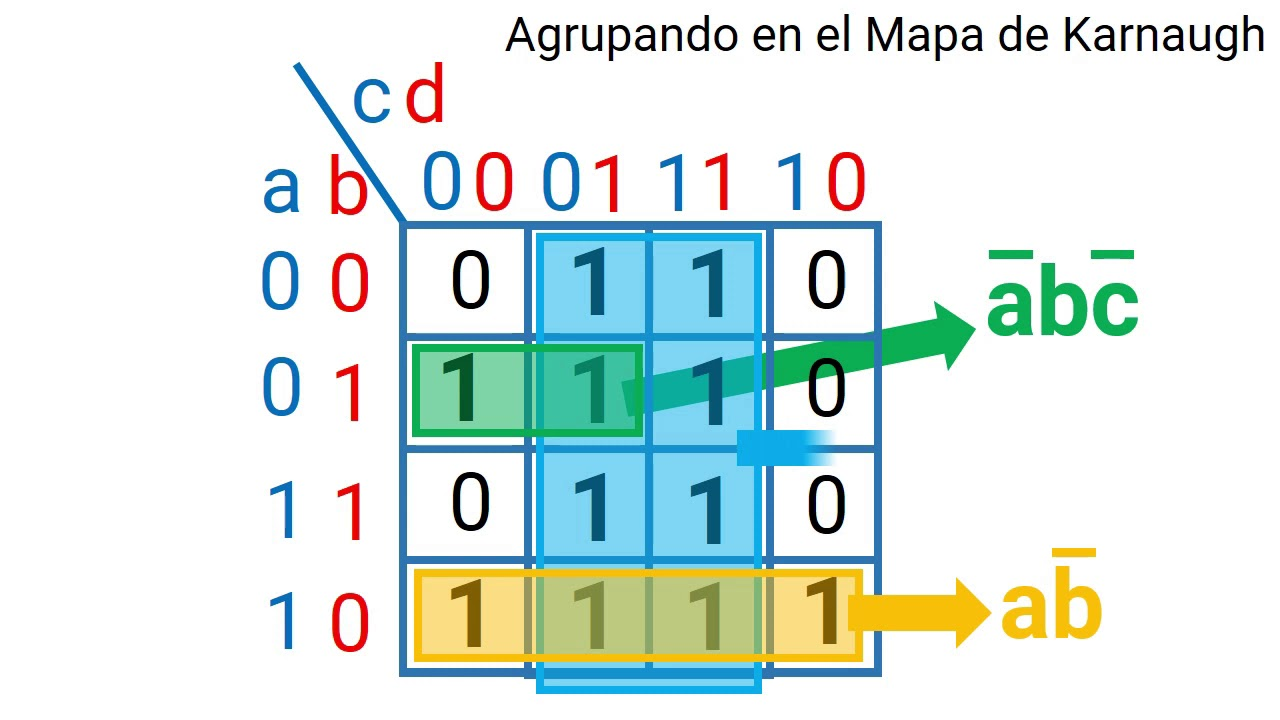
**V = W’Y’ + W’Z’** resultado final.

### **Implementaciones en Circuitos Lógicos.**

* 1. **Concepto:** Los circuitos lógicos son fundamentales en la electrónica digital y se utilizan para realizar operaciones lógicas y aritméticas basadas en señales digitales (representadas típicamente como 0s y 1s). Estos circuitos están compuestos por componentes electrónicos que implementan funciones booleanas, permitiendo la manipulación de información binaria de acuerdo con reglas lógicas predefinidas
  2. **Gráfico: **

### **Métodos de Simplificación: Mapas de Karnaugh**

* 1. **Concepto:** Un mapa de Karnaugh o mapa-K es una herramienta gráfica usada para simplificar una ecuación lógica o para convertir una tabla de verdad en su correspondiente circuito lógico. Aunque un mapa K puede usarse para resolver problemas con cualquier número de variables de entrada en la práctica se usa como máximo para cinco o seis variables.

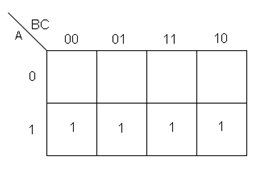


* 1. **Representación:** Los mapas de Karnaugh se organizan en una cuadrícula bidimensional, donde cada celda representa una combinación específica de variables de entrada del circuito. Las dimensiones de la cuadrícula dependen del número de variables en la función booleana que se está simplificando. Por ejemplo, una función booleana con dos variables tendría un mapa de Karnaugh de 2x2, mientras que una función con tres variables requeriría un mapa de 2x4.

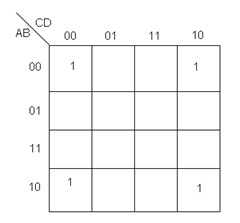
Las celdas dentro del mapa de Karnaugh están etiquetadas de manera

secuencial según la codificación binaria de las variables de entrada. Por ejemplo, en un mapa de Karnaugh de dos variables, las celdas se etiquetarían como 00, 01, 11, y 10, que corresponden a las combinaciones binarias posibles de las variables.

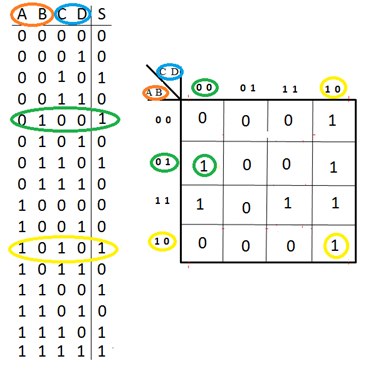
Mapa 2 X 4



Mapa de 4 x 4



* 1. Ejemplo:



Se muestra en la imagen que la tabla tiene 4 variables colocadas de la A a la D, ordenadas en columnas. Estas 4 variables forman diferentes combinaciones y cada combinación tiene una salida lógica {0, 1}, por ejemplo; la combinación mostrada en amarillo puede ser representada como, esta es la representación canónica, es decir, es una combinación que involucra a todas las variables y que tiene como salida un 1, hay que hacer notar que cuando el valor de la variable es 1, solo se representa con el nombre de la variable, sin embargo, cuando el valor de la variable es 0, se representa con la variable negada.

Como se aprecia en la imagen, las variables A y B, encerradas en anaranjado, son pasadas al mapa como si se tratara del eje de las Y de un plano cartesiano; las variables C y D son pasadas al mapa como si se tratara del eje de las X dentro del plano.

Ahora, para saber dónde colocar los valores vamos a utilizar la combinación enmarcada con verde, podemos ver que a las variables A y B les corresponden los valores 0 y 1 respectivamente, buscamos en el eje de las Y en donde se encuentra esa combinación de valores, y lo podemos ver también enmarcado con verde; para las variables C y D, se tienen los valores 0 y 0, por lo que, buscamos en el eje de las X tal combinación, la cual también es marcada con verde, ahora el valor que le corresponde, en este caso 1, se coloca en la casilla donde ambas combinaciones se intersectan. Este mismo procedimiento se puede ver para la combinación enmarcada en amarillo. Obviamente, todas las casillas del mapa deben de ser llenadas de la misma forma.

### **Conclusiones**

### **Referencias**

### 