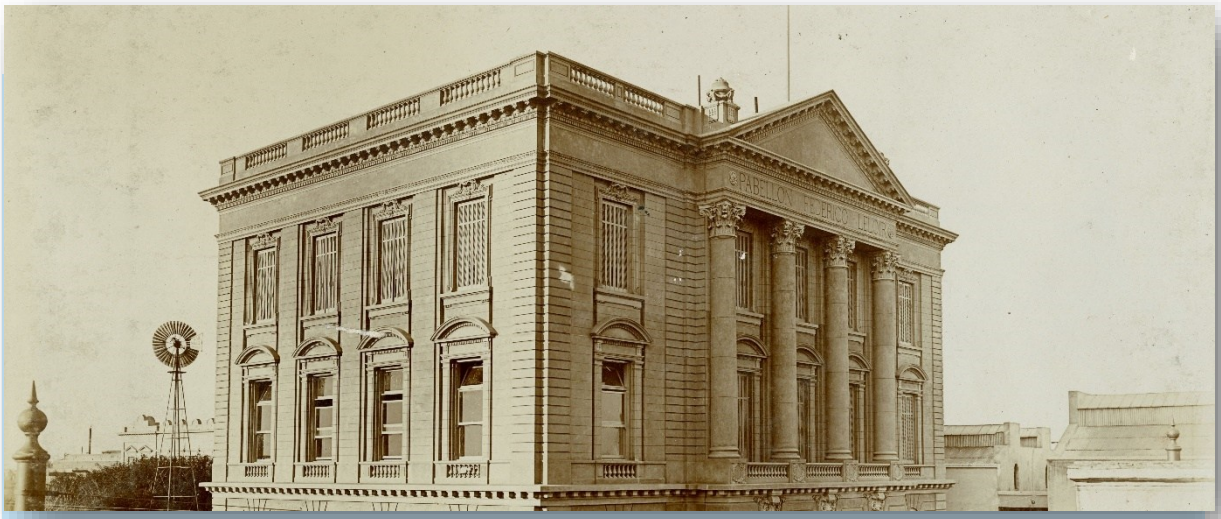
UTN

INGRESO 2025

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN A DISTANCIA

UNIVERSIDAD TECNOLÓGICA NACIONAL



CUADERNILLO 3

Introducción a la lógica

CURSO COMPLETO

UNIDAD I FUNDAMENTOS LOGICOMATEMÁTICOS

CUADERNILLO 1 – Teoría de conjuntos, números y sus tipos

CUADERNILLO 2 – Sistema Binario

CUADERNILLO 3 – Introducción a la lógica

CUADERNILLO 4 – Operaciones aritméticas

CUADERNILLO 5 – Números Enteros

CUADERNILLO 7 – Más de números

UNIDAD II RESOLUCIÓN DE PROBLEMAS

CUADERNILLO 6 – Análisis verbal

CUADERNILLO 8 – Método iterativo

CUADERNILLO 9 – Analogía y Patrones

CUADERNILLO 10 – Divide y conquistarás

CUADERNILLO 11 – Integración

CUADERNILLO 12 – Ensayo y Error

3: LOGICA

CONCEPTOS

¿QUE ES LA LOGICA?

La lógica es el estudio de proposiciones y su valor de verdad. Básicamente, te ayuda a pensar en forma ordenada y a deducir nueva información a partir de datos o conocimiento previo.

Hay dos tipos principales:

1) Lógica formal: Se enfoca en estructuras y reglas abstractas
2) Lógica aplicada: Busca resolver problemas concretos, como en la programación.

¿POR QUÉ LA ESTUDIAMOS?

Vamos a enfocarnos en la lógica formal, pero siempre con su aplicación en programación. La estudiamos para crear instrucciones claras que la compu entienda. Por ejemplo:

- Decidir si un número es par o impar
- Comprobar si un usuario pasa un filtro de seguridad.

“¿Es par el número 8?” → Respuesta lógica: “Verdadero, porque...”.

“¿3 es mayor que 7?” → Respuesta lógica: “Falso, porque...”.

LOGICA Y COMPUTACIÓN

En los lenguajes de programación la lógica es clave para evaluar CONDICIONES y tomar DECISIONES.

Por ejemplo,

SI (edad >= 18) { ... } decide si podés entrar a una web +18.

Relación entre algoritmos y lógica estructurada:

Un algoritmo es una secuencia de pasos lógicos. Si dominás la lógica, sabrás construir algoritmos más claros y eficientes.

HISTORIA BREVE

Desde la filosofía de la antigua Grecia hasta la actualidad, la lógica pasó de ser algo teórico a revolucionar la electrónica y la informática.

George Boole y el álgebra booleana:

Este genio matemático convirtió la lógica en operaciones matemáticas usando operadores lógicos AND, OR, NOT, sentando las bases de los circuitos lógicos.

Hardware y software:

Las puertas lógicas AND, OR, NOT son la base de los procesadores. El software usa esas mismas operaciones para funcionar.

¿Qué son las compuertas? Paciencia, ya llegaremos.

CONCEPTO DE PROPOSICIÓN

Una proposición es un enunciado que tiene un valor de verdad (Verdadero o Falso).

Ejemplos:

“5 es mayor que 3” es Verdadero (V)

“los números irracionales tienen decimales exactos” es Falso (F)

“2 + 2 = 4” es Verdadero (V)

"El 23 de mayo de 2789 Napoleón almorzó pollo" No sabemos si es Verdadero o Falso, pero si tiene un valor de verdad.

En cambio...

“Abre la puerta” → No tiene valor de verdad, es una orden.

A estos enunciados los llamamos PROPOSICIÓN SIMPLE porque tiene UN SOLO VALOR DE VERDAD.

PROPOSICIÓN COMPUESTA

Cuando combinamos dos o más proposiciones simples, obtenemos un PROPOSICION COMPUESTA.

Proposición 1: Tengo agua caliente

Proposición 2: Tengo yerba

Proposición compuesta: Tengo agua caliente (Y) tengo yerba

¿Es verdadera o falsa?

Proposición 1: Tengo huevos

Proposición 2: Tengo papas

Proposición 3: Tengo aceite

Proposición compuesta: Tengo huevo Y tengo papas (Y) tengo aceite

¿Es verdadera o falsa?

Proposición 1: Hoy llueve

Proposición 2: Hoy hace calor

Proposición compuesta: Hoy llueve (Y) hoy hace frio

¿Es verdadera o falsa?

Hemos unidos varias proposiciones con CONECTORES. El valor de verdad de la proposición compuesta DEPENDE de las SIMPLES y de la naturaleza del CONECTOR.

CONDICIÓN LÓGICA

En programación, una condición lógica es una proposición simple o compuesta que el código evalúa para decidir que hacer:

- Ejecuto un bloque de código si se cumple una condición.
- Repito algo mientras una condición se cumpla.

CONECTORES LOGICOS: AND, OR, NOT

Para entender cómo se evalúa una CONDICION LOGICA compuesta, se necesita conocer las formas de CONECTAR proposiciones más simples.

AND (Y)

Se DEBEN CUMPLIR TODAS las condiciones conectadas

Símbolos: Según el lenguaje se simboliza && o and

(a > 0 && b < 10): “tomo mate”

En lenguaje coloquial: "Si tenemos agua caliente **Y** yerba entonces tomamos mate"

Si no hay agua, o no hay yerba, ¿Es posible tomar mate?

OR (O):

Se DEBE CUMPLIR POR LO MENOS una condición

Símbolos: Según el lenguaje se simboliza || u or

(llueve || haceFrio): “me quedo en casa”

Sentencia que verifica que al menos una de ambas condiciones sea cierta, no necesariamente todas.

En lenguaje coloquial: "Si llueve **O** hace frío me quedo en casa".

Si ocurre una de ellas y no la otra, ¿Salgo o no?

NOT (NO):

Invierte el valor de verdad.

Símbolos: Según el lenguaje se simboliza: ! o not.

(!autorizado): “ingreso bloqueado”

Sentencia que verifica que no se cumpla la variable.

Lenguaje coloquial: "Si NO está autorizado, entonces el ingreso está bloqueado"

TABLA DE VERDAD

Las tablas de vedad son super útiles para determinar el valor de verdad de proposiciones complejas. Supongamos que tenemos dos proposiciones: p y q.

Estas tablas son cuadros donde listás todas las combinaciones posibles de valores (V/F) para las proposiciones y calculás el resultado final.

AND (&&, and) Ambas se deben cumplir		
p	**q**	**p and q**
F	F	F
F	V	F
V	F	F
V	V	V

Ejemplo: "manejo porque tengo licencia de conducir y hoy no tomé alcohol".

Si falla una de las condiciones, no manejo. Solo si se cumplen ambas puedo manejar.

OR (||, or) Al menos una se cumple		
p	**q**	**p and q**
F	F	F
F	V	V
V	F	V
V	V	V

Solo una combinación es falsa, si ambas no se cumplen.

Ejemplo: "Esta noche voy al cine o al recital".

Si se cumple una de las condiciones, significa que salí. Inclusive si se cumplen las dos. El único caso falso es cuando ninguna condición se cumple.

NOT (!, not) Ambas se deben cumplir	
p	**!p**
F	V
V	F

Cambia el valor de verdad.

Ejemplo: "No digo nada". Típica frase cuando queremos decir que no hablamos. Si la pensás tenemos "digo nada" es decir, no digo. Pero si lo niego diciendo "no", ¡en verdad si estoy diciendo algo!

XOR (^, xor) Solo si una se cumple		
p	**q**	**p xor q**
F	F	F
F	V	V
V	F	V
V	V	F

Solo se cumple si una combinación es verdadera, si ambas lo son no se cumplen.

Ejemplo: "Ese animal es vegetariano o carnívoro".

RELACION CON TEORIA DE CONJUNTOS

Ahora cambiamos el enfoque para conectar la lógica con la Teoría de Conjuntos.

¿Porque es importante?

Porque es la base del álgebra relacional, fundamental entender las consultas en bases de datos relacionales.

En lugar de proposiciones, ahora usamos conjuntos: A y B.

Si un elemento pertenece a un conjunto, lo indicamos en vez de poner Verdadero o Falso.

INTERSECCIÓN Debe estar en ambos conjuntos		
A	**B**	**A ∩ B**
no pertenece	no pertenece	no pertenece
no pertenece	pertenece	no pertenece
pertenece	no pertenece	no pertenece
pertenece	pertenece	pertenece

También se denomina CONJUNCIÓN LÓGICA.

UNION Todos incluyendo a los comunes		
A	**B**	**A ∪ B**
no pertenece	no pertenece	no pertenece
no pertenece	pertenece	pertenece
pertenece	no pertenece	pertenece
pertenece	pertenece	pertenece

Solo una combinación es falsa, si ambas no se cumplen.

COMPLEMENTO Ambas se deben cumplir	
A	**A'**
no pertenece	pertenece
pertenece	no pertenece

Básicamente si pertenece al A lo contrario es que no pertenezca. Lo mismo al revés.

DIFERENCIA SIMETRICA (XOR) No incluye a los comunes		
A	**B**	**A ⊕ B**
no pertenece	no pertenece	no pertenece
no pertenece	pertenece	pertenece
pertenece	no pertenece	pertenece
pertenece	pertenece	no pertenece

Tiene que pertenecer a ambos, no a los dos al mismo tiempo.

ANALISIS DE PROPOSICIONES COMPLEJAS

La mejor forma de analizar condiciones lógicas es hacer una tabla de verdad.

"(nota >= 6 and asistenciaClase) or (examenEspecial)"

PASO 1: Identifica cada proposición simple con una letra.

p = nota >=6

q = asistenciaClase

r = examenEspecial

PASO 2: Calculás cuantas filas te llevará. Como los valores de verdad son dos, es similar al binario. Por eso con 3 proposiciones es

2^3= 8 filas

PASO 3: Armas una columna por proposición, en este caso 3.

PASO 4: Luego una columna por operación que realizás. En este caso la del paréntesis, y otra para la final

PASO 5: Rellenás las posibilidades de valores de verdad y resolvés cada operación.

(p && q) || r				
p	**q**	**r**	**p && q**	**(p && q) || r**
F	F	F	F	F
F	F	V	F	V
F	V	F	F	F
F	V	V	F	V
V	F	F	F	F
V	F	V	F	V
V	V	F	V	V
V	V	V	V	V

Nos damos cuenta qué esta condición compuesta se cumple, excepto en 3 casos.

Lo interesante de este análisis es que, si el resultado siempre es falso, no importando el valor de verdad de las condiciones simples, NUNCA SE EJECUTARÁ EL CÓDIGO.

Por el contrario, si siempre es VERDAD, no sirve como condición, el CÓDIGO SIEMPRE SE EJECUTARÁ.

INTRO AL ALGEBRA DE BOOLE

VARIABLES BOOLEANAS

Variables que solo pueden tomar los valores Verdadero (True) o Falso (False).

USO PRACTICO: Guardar resultados de comparaciones, status de usuario, flags de estado en un juego, etc.

BOOLEANOS Y BITS

Cuando pensamos los valores de verdad nos damos cuenta de que son solo 2, false o true. Cada uno de ellos puede representarse con solo 1 bit:

TRUE: 1

FALSE: 0

OPERACIONES BOOLEANAS

Las variables las podemos sumar o multiplicar.

Recordá estas reglas:

1) puedo sumar y se comporta como un OR o UNION
2) puedo multiplicar y se comportan como un AND o INTERSECCION
3) También puedo negarlas

Ejemplo 1:

A y B son variables booleanas

A es TRUE, B es FALSE

A + B = TRUE (porque se comporta como una unión u or)

Ejemplo 2:

A y B son variables booleanas

A es TRUE, B es FALSE

A * B = FALSE (porque se comporta como una intersección o AND)

O SEA, ESTAMOS OPERANDO COMO SI FUERAN NÚMEROS, A VARIABLES BOOLEANAS. ESTO ES ALGEBRA DE BOOLE

Las leyes y otras operaciones complejas las vemos en matemática.

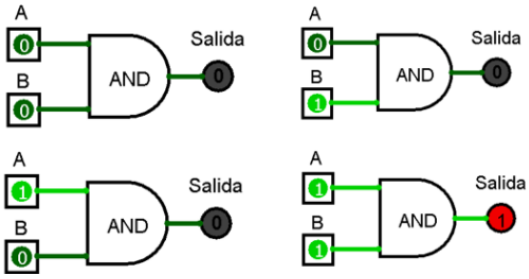
TABLAS DE VERDAD, BITS, COMPUERTAS

La lógica booleana reemplaza los valores de verdad (V o F) en su equivalente en un bit (1 o 0).

Cada bit indica si hay corriente o no.

En los circuitos electrónicos, existen estructuras microscópicas llamadas transistores, que se combinan para formar compuerta electrónica.

Estas compuertas deciden si deja pasar o no corriente según las señales de entrada.

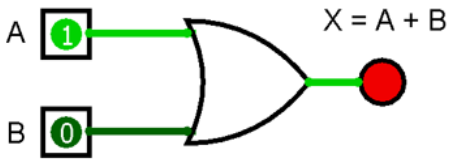


Entradas | Salida

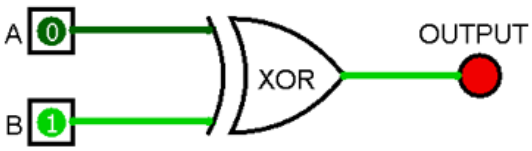
A	B	Salida
0	0	0
0	1	0
1	0	0
1	1	1

COMPUERTA AND Ambas entradas deben estar activas		
Entrada A	**Entrada B**	**Salida**
0	0	0
0	1	0
1	0	0
1	1	1

A	B	X= A + B
0	0	0
0	1	1
1	0	1
1	1	1



COMPUERTA OR Al menos una entrada debe estar activa		
Entrada A	**Entrada B**	**salida**
0	0	0
0	1	1
1	0	1
1	1	1

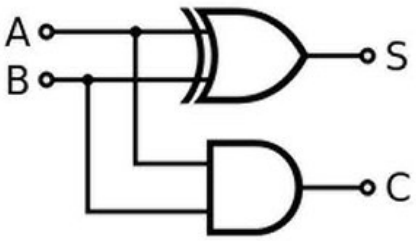


A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	0

COMPUERTA XOR Al menos una entrada debe estar activa		
Entrada A	**Entrada B**	**salida**
0	0	0
0	1	1
1	0	1
1	1	1

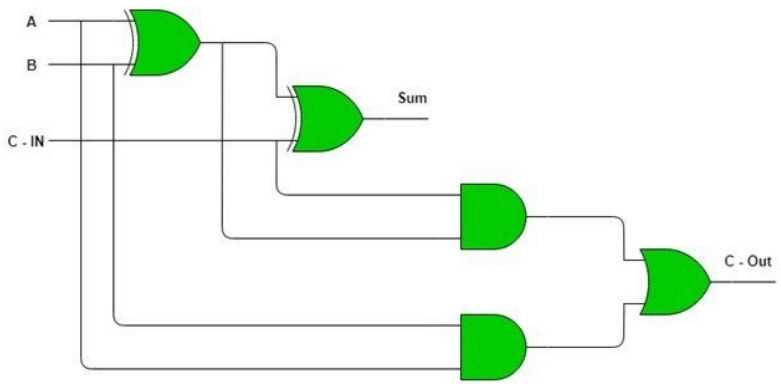
Hay más, pero lo importante es que entiendas como relacionamos lógica con electrónica y, por ende, nuestras computadoras.

Así si sumamos básicamente tenemos dos bits de entrada y necesitamos que sea 1 si solo uno de ellos es 1 (compuerta XOR) y si ambos son 1 no solo debe indicarse como 0 (compuerta XOR), además indicar que hay un 1 como carry (compuerta AND).



Así con lógica pasamos al álgebra de Boole, de ahí a la electrónica y por último llegar a sumar con una computadora.

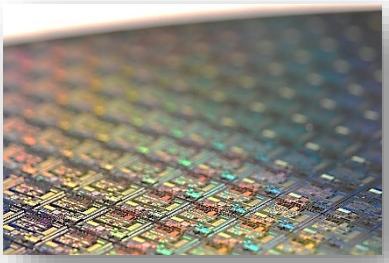
Por si te interesa así manejamos electrónicamente el carry en el siguiente bit. A este circuito se llega mediante diseño usando el álgebra de Boole, pero lo dejamos para las cursadas del primer nivel.



TRANSISTORES Y COMPUERTAS

Tamaño:

Es tan grande el avance en miniaturización que un procesador Intel i7 de 12° generación puede contener en 215 mm^2 entre 20 y 21 mil millones de transistores. Por eso hablamos de estructuras microscópicas.



Usos:

Las vemos en cantidades en las CPU, GPU, TPU (para machine learning), NPU (unidad de procesamiento neural), DSP (usados en procesamientos de señales), DPU (Para procesamiento de grandes cantidades de datos). En las memorias RAM, en los discos de estado sólido (flash), etc.

BUGS

En las primeras computadoras electrónicas los transistores eran del tamaño de lámparas.

Generaban gran cantidad de calor que atraían a bichos. Cuenta la historia que la Mark II empezó a registrar errores. Una pionera de la informática llamada Grace Hooper encontró una polilla entre dos relés. A partir de ese momento en informática asociamos el término bugs (bicho) a errores del software.

VALIDACIÓN DE ENTRADAS CON LOGICA

Verificar si una contraseña cumple criterios (mínimo 8 caracteres, un número, un símbolo).

tieneSimbolo && tieneDigito && longitud >= 8

Esto se usa para controlar accesos, formularios, etc.

RESOLUCION DE PROBLEMAS TIPICOS

PROBLEMA: “Calcular si un año es bisiesto”

Regla lógica: “Si es divisible por 4 y no por 100, o si es divisible por 400”

Traducimos en una condición:

(year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

Nota: "==" es igual, y "!=" es distinto.

PROBLEMA: “Verificar si un número pertenece a un rango”

Regla lógica: “El valor x debe estar en el intervalo 1 al 100”

Traducimos en una condición:

(x >= 1 && x <= 100)

La lógica booleana hace que tu código refleje esas reglas exactas sin ambigüedades.