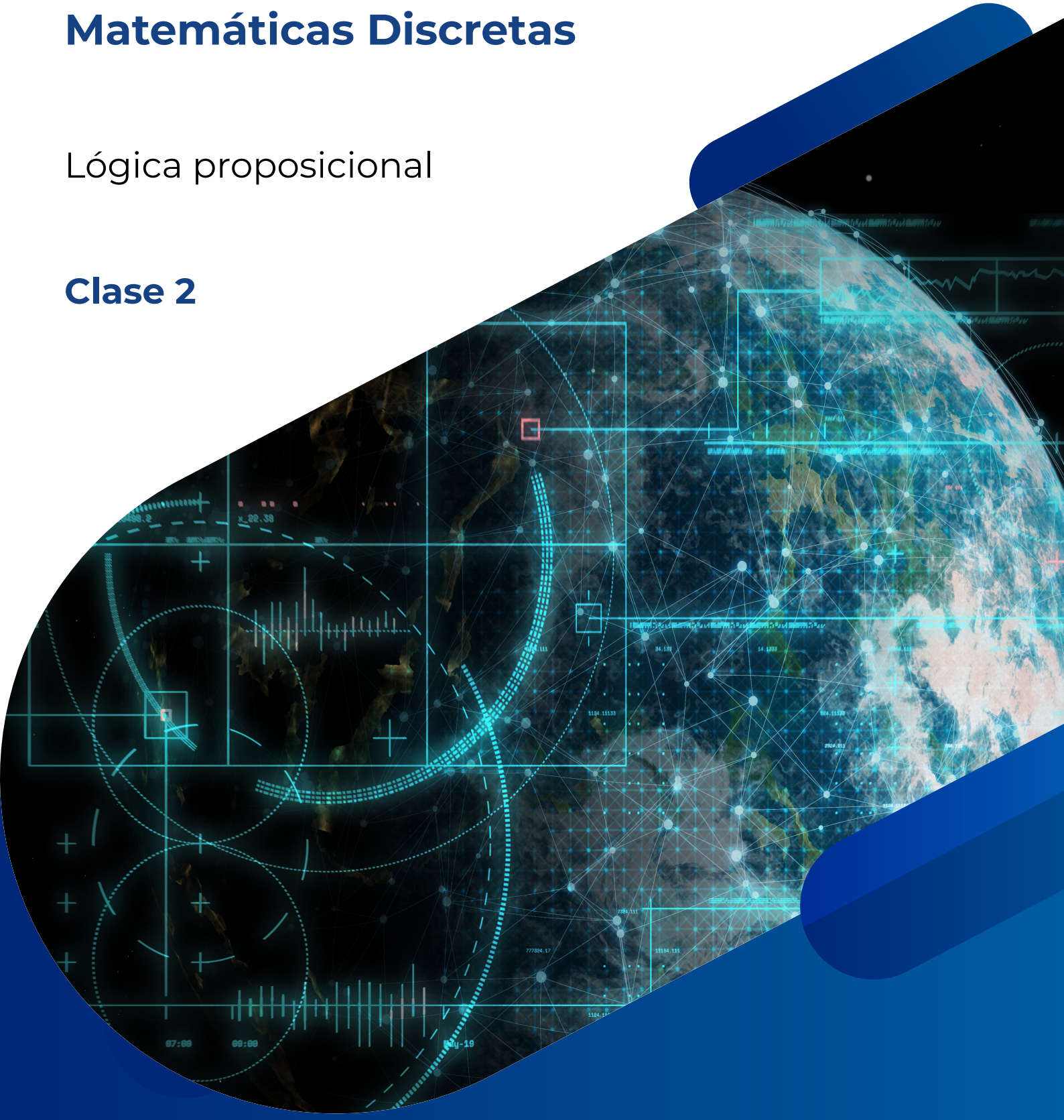
Matemáticas Discretas

Lógica proposicional

Clase 2



La excelencia no se improvisa

INTRODUCCIÓN DE LA CLASE

¿Cómo sabemos si una proposición es verdadera o falsa? A veces, esta pregunta puede parecer subjetiva o depender del contexto. Sin embargo, en lógica matemática nos enfocamos en una forma precisa de responderla: si conocemos el valor de verdad de ciertas proposiciones simples, ¿podemos determinar el valor de verdad de una nueva proposición que se construye a partir de ellas? Este es justamente el corazón de la clase: aprenderemos a evaluar proposiciones compuestas a partir de otras más simples, utilizando herramientas formales como las tablas de verdad.

A lo largo de esta sesión, aprenderás a construir y analizar tablas de verdad para distintos conectivos lógicos, identificar cuándo una proposición es siempre verdadera (tautología), siempre falsa (contradicción) o equivalente a otra en estructura y significado. También exploraremos el concepto de formas normales, que permiten reescribir proposiciones utilizando conectivos suficientes de manera sistemática. Al finalizar, habrás desarrollado habilidades para evaluar argumentos lógicos con rigor, un paso fundamental para aplicaciones en programación, seguridad informática y razonamiento formal.

Clase 2: Lógica proposicional

RDA 2: Resolver problemas prácticos y teóricos mediante la aplicación de técnicas y herramientas de la Matemática Discreta, como lógica matemática, análisis de conjuntos y teoría de grafos en el manejo de estructuras discretas.

2. Tablas de Verdad

¿Alguna vez te has preguntado cómo una computadora decide qué datos mostrarte cuando filtras por más de una condición? Por ejemplo, en una hoja de cálculo puedes aplicar un filtro para mostrar únicamente los registros donde el campo «Acceso permitido» sea verdadero y el campo «Intentos fallidos» sea menor a 3. En este caso, estás utilizando una combinación de condiciones que, en el fondo, responde a la lógica proposicional.

Para poder analizar con claridad este tipo de situaciones, introducimos una herramienta fundamental: **las tablas de verdad**. Estas nos permiten observar qué valor (verdadero o falso) adopta una proposición compuesta según los valores de sus componentes más simples (Epp 2020). A este proceso lo llamamos una asignación de valores de verdad: consiste en suponer que cada proposición simple tiene un valor definido (verdadero o falso) y, a partir de ello, calcular el valor de verdad de la proposición completa. Así, podemos explorar todas las combinaciones posibles y conocer el comportamiento lógico completo de cualquier expresión.

Retomemos el ejemplo de filtrar registros en una base de datos. Supongamos que queremos mostrar únicamente aquellos registros en los que se cumplen las siguientes dos condiciones:

- Que el usuario tenga acceso permitido (p).
- Que el número de intentos fallidos sea menor que 3 (q).

Queremos evaluar si se cumplen ambas condiciones al mismo tiempo, es decir, si se cumple la proposición compuesta $p \wedge q$.

Antes de continuar, adoptaremos una convención: en lugar de escribir «verdadero» o «falso» en la tabla, utilizaremos **V** y **F**, respectivamente.

Ahora sí, construiremos la tabla de verdad considerando todas las asignaciones posibles de valores de verdad para p y q, y evaluaremos el valor resultante de $p \wedge q$ en cada caso:

- Si el valor asignado a p es **V** (el usuario tiene acceso permitido), y el valor asignado a q también es **V** (menos de 3 intentos fallidos), entonces ambas condiciones se cumplen, por lo que $p \wedge q$ también es **V**.
- Si el valor asignado a p es **V**, pero el valor asignado a q es **F** (3 o más intentos fallidos), no se cumplen ambas condiciones, así que $p \wedge q$ es **F**.
- Si el valor asignado a p es **F** (el acceso no fue permitido), aunque el valor asignado a q sea **V**, la condición conjunta no se cumple: $p \wedge q$ es **F**.
- Finalmente, si ambos valores asignados son **F**, claramente $p \wedge q$ también es **F**.

Esto se resume en la siguiente tabla:

Tabla 1. Tabla de verdad de $p \wedge q$

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

Nota: tomado de Epp (2020).

Observa que esta es la **tabla de verdad de la proposición** $p \wedge q$, sin importar lo que representen exactamente las proposiciones p y q. Siempre que analicemos una conjunción, este será su comportamiento lógico.

Ahora, supón que queremos hacer otro filtrado de registros, pero esta vez con un criterio más flexible. Imaginemos que deseamos mostrar todos los registros que cumplan al menos una de las siguientes condiciones:

- p: «El usuario tiene acceso permitido»
- q: «El número de intentos fallidos es menor que 3»

Esto significa que aceptaremos registros en los que el acceso esté permitido, o bien donde el número de intentos fallidos no supere el límite, o incluso ambos casos a la vez. En términos lógicos, esto se expresa mediante la proposición $p \vee q$.

Te invito a construir la tabla de verdad para $p \vee q$, considerando todas de la misma manera que lo hicimos para $p \wedge q$. El resultado debería ser:

Tabla 2. Tabla de verdad de $p \vee q$

p	q	$p \wedge q$
V	V	V
V	F	V
F	V	V
F	F	F

Nota: tomado de Epp (2020).

2.1. Función de verdad

Una forma sencilla de trabajar con valores de verdad es codificarlos como números binarios: escribiremos **1** para «verdadero» y **0** para «falso». Adoptada esta convención, la tabla de verdad de la conjunción $p \wedge q$ queda:

Tabla 3

Tabla de verdad de $p \wedge q$ en notación binaria

p	q	$p \wedge q$
1	1	1
1	0	0
0	1	0
0	0	0

Nota: adaptado de Epp (2020).

Esto nos permite formalizar el significado de una asignación de valores de verdad. Si tenemos dos letras proposicionales, una asignación consiste en elegir un valor de verdad (1 o 0) para cada una. Por ejemplo, la asignación $(1,1)$ indica que el valor asignado a p es 1 (verdadero) y el valor asignado a q también es 1. Bajo esta asignación, el valor de verdad de $p \wedge q$ es 1.

Definición 1: Asignación de valores de verdad.

Sea $n \in \mathbb{N}$. Una *asignación de valores de verdad* para n letras proposicionales es un elemento del conjunto $\{0,1\}^n$, es decir, una n-tupla de ceros y unos que determina el valor de verdad (0 o 1) de cada proposición elemental.

Con esto, para cada asignación de valores de verdad de p y q se tiene un valor de verdad de $p \wedge q$:

Tabla 4

Asignaciones y valor de $p \wedge q$

Asignación (p, q)	**Valor de verdad de** $p \wedge q$
(1,1)	1

Asignación (p, q)	**Valor de verdad de** $p \wedge q$
(1,0)	0
(0,1)	0
(0,0)	0

Nota: creación propia (Merino, A., 2025).

De esta manera, podemos ver que, la tabla de verdad no es más que una función

$$f: \{0,1\}^2 \to \{0,1\}.$$

De manera general, tenemos la siguiente definición:

Definición 2: Función de verdad.

Para $n \in \mathbb{N}$, una *función de verdad* de n variables es una función de la forma

$$f: \{0,1\}^n \to \{0,1\}.$$

Es decir, una regla que asigna a cada combinación posible de valores de entrada (ceros y unos) un único valor de salida (0 o 1).

Una vez formalizados estos conceptos, a continuación, se presentan las tablas de verdad de los conectores lógicos estudiados en la clase anterior:

Tabla 5

Valores de verdad para varios conectores lógicos

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \to q$	$p \leftrightarrow q$
1	1	0	1	1	1	1
1	0	0	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1

Nota: adaptado de Epp (2020).

Con esta base, podemos construir tablas de verdad de proposiciones más complejas.

2.2. Evaluación de conectivos

Para construir una tabla de verdad de una proposición lógica compuesta, se procede de la siguiente manera:

1. Se identifica el número n de letras proposicionales distintas que intervienen.
2. Se listan todas las posibles asignaciones de valores de verdad para dichas proposiciones. Como cada letra puede tomar dos valores (0 o 1), el número total de combinaciones posibles es 2^n, es decir, la tabla tendrá 2^n filas.
3. Se organiza la tabla de forma que las columnas correspondientes a las letras proposicionales reflejen todas las combinaciones en orden sistemático:
   - A la primera letra se le asignan 2^{n-1} valores verdaderos (la mitad de filas) seguidos de 2^{n-1} valores falsos (la otra mitad de filas).
   - A la segunda, 2^{n-2} verdaderos (la mitad de filas del paso anterior), 2^{n-2} falsos, repitiendo ese patrón.
   - Y así sucesivamente, dividiendo la secuencia por mitades.
4. Finalmente, se completan las columnas adicionales con los valores de verdad de las proposiciones compuestas, hasta obtener el valor de la proposición final.

Ejemplo:

Tabla de verdad de $(p \vee q) \rightarrow q$

Vamos a construir la tabla de verdad para la proposición lógica compuesta $(p \vee q) \rightarrow q$, siguiendo los pasos indicados.

1. **Identificación de letras proposicionales:** Las letras que aparecen son p y q. Por tanto, $n = 2$.
2. **Número de combinaciones:** El número de combinaciones posibles de valores de verdad es $2^n = 2^2 = 4$. La tabla tendrá 4 filas.
3. **Asignación sistemática de valores de verdad:**

p	q
1	1
1	0
0	1
0	0

4. **Evaluación de las proposiciones compuestas:**

   – Calculemos $p \vee q$: la disyunción es verdadera cuando al menos una de las proposiciones lo es.

   – Luego, evaluamos $(p \vee q) \rightarrow q$: una implicación es falsa solo cuando el antecedente es verdadero y el consecuente es falso.

Tabla 6

Tabla de verdad de $(p \vee q) \rightarrow q$

p	q	$p \vee q$	$(p \vee q) \rightarrow q$
1	1	1	1
1	0	1	0
0	1	1	1
0	0	0	1

Nota: creación propia (Merino, A., 2025).

Para proposiciones más complejas, el procedimiento para construir tablas de verdad sigue exactamente el mismo lineamiento. Por ejemplo, en la siguiente tabla se evalúa la expresión compuesta:

$$(p \vee \neg q) \wedge r.$$

Dado que hay tres letras proposicionales, el número de combinaciones será $2^3 = 8$, es decir, la tabla tendrá 8 filas. Siguiendo la secuencia lógica: primero se calcula $\neg q$, luego $p \vee \neg q$, y finalmente se evalúa la conjunción completa con r.

Tabla 7

Tabla de verdad de $(p \vee \neg q) \wedge r$ *en notación binaria*

p	q	r	$\neg q$	$p \vee \neg q$	$(p \vee \neg q) \wedge r$
1	1	1	0	1	1
1	1	0	0	1	0
1	0	1	1	1	1
1	0	0	1	1	0
0	1	1	0	0	0
0	1	0	0	0	0
0	0	1	1	1	1
0	0	0	1	1	0

Nota: creación propia (Merino, A., 2025).

Para ver un ejercicio de mayor complejidad resuelto a detalle, se recomienda revisar el siguiente video, donde se construye paso a paso una tabla de verdad para una proposición compuesta:

- **Título del enlace:** Tablas de verdad | Ejemplo
- **Descripción del enlace:** El video desarrolla paso a paso una tabla de verdad compleja.
- **Enlace:** https://youtu.be/pcgs0FRpnbQ?si=wOHqUNy9dl9ouFOc

Generación con Python:

La construcción manual de tablas de verdad es una excelente forma de comprender el proceso lógico. Sin embargo, cuando se trabaja con fórmulas más largas o con más variables, puede resultar útil automatizar este procedimiento. Una forma sencilla de hacerlo es utilizando el lenguaje de programación Python junto con la biblioteca ttg (truth-table-generator 2025).

El siguiente código genera automáticamente la tabla de verdad de la proposición $(p \vee \neg q) \wedge r$, junto con algunas subexpresiones intermedias:

Código 1

Generación de Tabla de Verdad con ttg

```
from ttg import Truths

# Creamos una tabla de verdad con las variables p, q y r
# y tres fórmulas: ~p, p or ~q, (p or ~q) and r
tabla = Truths(['p', 'q', 'r'], ['~p', 'p or ~q', '(p or ~q) and r'])

# Mostramos la tabla generada
print(tabla)
```

Este código realiza automáticamente las siguientes acciones:

- Genera todas las combinaciones posibles de valores de verdad para las letras proposicionales p, q y r (en este caso, 8 combinaciones en total).
- Evalúa las tres expresiones dadas: la negación $\neg p$, la disyunción $p \vee \neg q$, y finalmente la conjunción completa $(p \vee \neg q) \wedge r$.
- Presenta la tabla completa con una columna para cada variable, cada subexpresión y el resultado final, facilitando la interpretación paso a paso del razonamiento lógico.

Esta herramienta es especialmente útil para validar tablas de verdad complejas, comprobar equivalencias, o preparar material educativo de forma rápida y confiable.

En la siguiente figura se muestra la salida obtenida, donde puede observarse la evaluación de cada una de las expresiones incluidas:

Figura 1

Salida de la ejecución en Python del generador de tablas de verdad

```
from ttg import Truths
print(Truths(['p', 'q', 'r'], ['~p', 'p or ~q', '(p or ~q) and r']))
```

```
+-----+-----+-----+------+-----------+-------------------+
|  p  |  q  |  r  |  ~p  |  p or ~q  |  (p or ~q) and r  |
|-----+-----+-----+------+-----------+-------------------|
|  1  |  1  |  1  |  0   |     1     |         1         |
|  1  |  1  |  0  |  0   |     1     |         0         |
|  1  |  0  |  1  |  0   |     1     |         1         |
|  1  |  0  |  0  |  0   |     1     |         0         |
|  0  |  1  |  1  |  1   |     0     |         0         |
|  0  |  1  |  0  |  1   |     0     |         0         |
|  0  |  0  |  1  |  1   |     1     |         1         |
|  0  |  0  |  0  |  1   |     1     |         0         |
+-----+-----+-----+------+-----------+-------------------+
```

Nota: creación propia (Merino, A., 2025).

2.3. Tautologías y contradicciones

Al analizar proposiciones compuestas, uno de los objetivos es determinar si su valor de verdad depende de la asignación o si, por el contrario, siempre resulta igual. En este sentido, se distinguen tres casos:

- Una proposición es una **tautología** si su valor de verdad es *siempre 1*, sin importar qué valores tomen las letras proposicionales.
- Una proposición es una **contradicción** si su valor de verdad es *siempre 0*, sin importar qué valores tomen las letras proposicionales.
- Una proposición es **contingente** si en algunas asignaciones resulta verdadera y en otras, falsa.

Estos conceptos son importantes porque nos permiten clasificar proposiciones según su comportamiento lógico, sin necesidad de conocer su contenido específico. Por ejemplo, una tautología puede interpretarse como una regla lógica válida, mientras que una contradicción señala un error estructural en el razonamiento.

Ejemplos:

1. $p \vee \neg p$ es una tautología. Para cualquier valor que tome p, la disyunción será siempre verdadera.
2. $p \wedge \neg p$ es una contradicción. Nunca puede ser verdadera porque no existe una asignación en la que p y $\neg p$ sean simultáneamente verdaderas.

3. $p \to q$ es una proposición contingente: su valor depende de la asignación particular de p y q.

A continuación, se muestra una tabla de verdad para la proposición $p \vee \neg p$, que permite verificar que su valor de verdad es siempre 1:

Tabla 8

Tabla de verdad de $p \vee \neg p$ (tautología) y de $p \wedge \neg p$ (contradicción)

p	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
1	0	1	0
0	1	1	0

Nota: creación propia (Merino, A., 2025).

2.4. Equivalencia lógica

En muchas ocasiones, dos proposiciones pueden parecer distintas a primera vista, pero al analizarlas con una tabla de verdad, descubrimos que se comportan de forma idéntica bajo todas las asignaciones posibles de valores. Cuando esto ocurre, decimos que las proposiciones son **lógicamente equivalentes**.

Es decir, dos formas proposicionales son equivalentes si tienen la misma función de verdad. En este caso, escribimos $P \equiv Q$. Esta noción de equivalencia es especialmente útil en programación, por ejemplo, al simplificar condicionales dentro de un if, donde una forma más corta o clara puede hacer el código más legible y eficiente.

Ejemplo 1 - Leyes de De Morgan:

Las leyes de De Morgan son equivalencias clásicas que permiten transformar negaciones de conjunciones o disyunciones. Estas son:

$$\neg(p \wedge q) \equiv \neg p \vee \neg q \qquad \text{y} \qquad \neg(p \vee q) \equiv \neg p \wedge \neg q.$$

Verifiquemos una de estas equivalencias con su tabla de verdad:

Tabla 9

Tabla de verdad de De Morgan: $\neg(p \wedge q) \equiv \neg p \vee \neg q$

p	q	$p \wedge q$	$\neg(p \wedge q)$	$\neg p$	$\neg p \vee \neg q$
1	1	1	0	0	0
1	0	0	1	0	1
0	1	0	1	1	1
0	0	0	1	1	1

Nota: creación propia (Merino, A., 2025).

Como se puede observar, las columnas de $\neg(p \wedge q)$ y $\neg p \vee \neg q$ coinciden completamente: son equivalentes.

Ejemplo 2 - Reescribir una implicación:

Otra equivalencia fundamental es la siguiente

$$p \rightarrow q \equiv \neg p \vee q.$$

Esta equivalencia es especialmente útil en programación para reescribir condicionales, por ejemplo:

if (p) then q es equivalente a if (not p or q).

2.4.1. Expresiones mínimas

Toda proposición lógica puede escribirse de múltiples formas equivalentes. Algunas de ellas pueden ser más simples que otras, dependiendo de los conectores que se utilicen. En lógica formal, existen métodos para encontrar estas formas equivalentes mínimas o «formas normales».

Por ejemplo, la siguiente proposición:

$$(p \rightarrow q) \vee (r \wedge \neg q)$$

es lógicamente equivalente a:

$$\neg p \vee q \vee r.$$

Puedes verificar esta equivalencia automáticamente con la siguiente herramienta de WolframAlpha, que genera la forma mínima y otras equivalencias posibles:

- **Título del enlace:** Forma normal de una proposición lógica
- **Descripción del enlace:** Introduce una proposición lógica y obtén formas equivalentes simplificadas, incluyendo su representación en forma normal.
- **Enlace:** https://www.wolframalpha.com/input?i=%28p+IMPLIES+q%29+OR+%28r+AND+%7Eq%29&lang=es

En lógica, no todos los conjuntos de conectores permiten expresar cualquier proposición posible. Por esta razón, se habla de conjuntos adecuados de conectores, es decir, colecciones de conectivos que son suficientes para construir cualquier función de verdad, sin necesidad de recurrir a conectores adicionales.

Un ejemplo clásico de conjunto adecuado es $\{\wedge, \vee, \neg\}$. Esto significa que cualquier proposición lógica—por más compleja que sea—puede escribirse utilizando únicamente conjunciones, disyunciones y negaciones.

Por ejemplo, los siguientes conectores pueden reescribirse así:

- Implicación:

$$p \rightarrow q \equiv \neg p \vee q.$$

- Bicondicional:

$$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q).$$

Este hecho tiene una aplicación directa en programación: basta con conocer y usar adecuadamente los conectores AND, OR y NOT para poder expresar cualquier condición lógica que deba ser evaluada dentro de estructuras como if, while o validaciones booleanas. Por ejemplo, en lugar de escribir una implicación, puede implementarse como una disyunción negada, haciendo el código más compatible con el lenguaje de programación en uso.

Además, en áreas como el diseño de circuitos digitales, la optimización de expresiones booleanas o la verificación de lógica en bases de datos, esta propiedad resulta esencial,

ya que permite trabajar siempre con un subconjunto reducido y eficiente de conectores lógicos.

Referencias

Epp, S. S. (2020). Discrete Mathematics with Applications (5.a ed.). Cengage Learning.

truth-table-generator. (2025). PyPI [Recuperado el 20 de julio de 2025]. https://pypi.org/project/truth-table-generator/

Términos

- **Tautología:** es una proposición que resulta verdadera para toda asignación posible de valores de verdad.
- **Lógicamente equivalentes:** dos proposiciones son lógicamente equivalentes si tienen el mismo valor de verdad bajo todas las posibles asignaciones.

Profundización

Recurso 1

- **Título del recurso:** Proposiciones y Tablas de Verdad en Python
- **Descripción del enlace:** Este recurso presenta una guía de Python para escribir proposiciones y tablas de verdad.
- **Enlace:** https://github.com/FCENA-PUCE/Python-Notebooks-Educativos/blob/main/Tablas-de-verdad.ipynb

PUCE Virtual

La excelencia no se improvisa

síguenos