Lógica Proposicional

Definición

- Una **proposición** es una afirmación con valor declarativo/informativo que puede ser verdadera (V) o falsa (F), pero no ambas cosas a la vez.

Ejemplos

- Expresiones que son proposiciones:
  - Hoy es viernes.
  - 5 > 25.
  - 7 es primo.
- Expresiones que no son proposiciones:
  - Hola.
  - ¿Cómo estás?
  - Quédense quietos.
  - ¡Buenísimo!

Tablas de verdad básicas

Negación $(\neg)$

p	$\neg p$
V	F
F	V

Conjunción $(\wedge)$

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

p	q	$p \wedge q$
F	F	F

Disyunción $(\vee)$

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

Condicional $(\rightarrow)$

p	q	$p \rightarrow q$
V	V	V
V	F	F
F	V	V
F	F	V

Bicondicional $(\leftrightarrow)$

p	q	$p \leftrightarrow q$
V	V	V
V	F	F
F	V	F
F	F	V

Equivalencia lógica

Definición

- Dos proposiciones p y q son lógicamente equivalentes si coinciden sus resultados para los mismos valores de verdad.
- Se denota $p \Leftrightarrow q$.

Ejemplos

- $p \to q \not\Leftrightarrow q \to p$
- $p \to q \not\Leftrightarrow \neg q \to \neg p$

Tautología

Definición

- Una tautología es una proposición que es verdadera para todos los valores de verdad de sus variables.
- Si $p \leftrightarrow q$ es una tautología, entonces $p \Leftrightarrow q$, es decir, p es lógicamente equivalente a q.

Ejemplos

- $(p \to q) \leftrightarrow (\neg q \to \neg p)$

Ejemplos de equivalencias conocidas

Doble negación

- $\neg\neg p \Leftrightarrow p$

Leyes conmutativas

- **De la conjunción**: $(p \wedge q) \Leftrightarrow (q \wedge p)$
- **De la disyunción**: $(p \vee q) \Leftrightarrow (q \vee p)$

Leyes asociativas

- **De la conjunción**: $((p \wedge q) \wedge r) \Leftrightarrow (p \wedge (q \wedge r)) \Leftrightarrow (p \wedge q \wedge r)$
- **De la disyunción**: $((p \vee q) \vee r) \Leftrightarrow (p \vee (q \vee r)) \Leftrightarrow (p \vee q \vee r)$

Leyes distributivas

- **De la conjunción**: $(p \vee (q \wedge r)) \Leftrightarrow ((p \vee q) \wedge (p \vee r))$
- **De la disyunción**: $(p \wedge (q \vee r)) \Leftrightarrow ((p \wedge q) \vee (p \wedge r))$

Leyes de idempotencia

- $p \wedge p \Leftrightarrow p$
- $p \vee p \Leftrightarrow p$

Leyes de De Morgan

- $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$
  - Se generaliza a n conjunciones: $\neg(A_1 \wedge A_2 \wedge ... \wedge A_n) \Leftrightarrow (\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_n)$
- $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$
  - Se generaliza a n disyunciones: $\neg(A_1 \vee A_2 \vee ... \vee A_n) \Leftrightarrow (\neg A_1 \wedge \neg A_2 \wedge ... \wedge \neg A_n)$

Contrapositiva

- $(p \to q) \Leftrightarrow (\neg q \to \neg p)$

Condicional

- $(p \to q) \Leftrightarrow \neg p \vee q$
- $(p \to q) \Leftrightarrow \neg(p \wedge \neg q)$

Bicondicional

- $(p \leftrightarrow q) \Leftrightarrow ((p \to q) \wedge (q \to p))$

Adición

- $p \implies (p \vee q)$

Simplificación

- $(p \wedge q) \implies p$

Modus Ponens

- $p \wedge (p \to q) \implies q$

Modus Tollens

- $\neg q \land (p \to q) \implies \neg p$

Transitividad del condicional

- $(p \to q) \land (q \to r) \implies (p \to r)$

Transitividad del bicondicional

- $(p \leftrightarrow q) \land (q \leftrightarrow r) \implies (p \leftrightarrow r)$

Esquemas Proposicionales

Definición

- Un esquema proposicional en la indeterminada x es una expresión que contiene a x y posee la propiedad: "Existe por lo menos un nombre tal que la expresión obtenida al reemplazar x por dicho nombre es una proposición válida".
- Las indeterminadas suelen llamarse variables o incógnitas.
- Ejemplo: "x es blanca" es un esquema, porque si reemplazamos x por "la flor" obtenemos la proposición "La flor es blanca".
- Normalmente se usan símbolos como $P(x)$, $Q(x)$, $F(x)$ para denotar esquemas de incógnita x.

Operadores universal y existencial

- Si se tiene un esquema $P(n)$ puede obtenerse a partir de él una proposición mediante el uso de los operadores:
  - **Universal**: $(\forall n) : (P(n))$
    - La proposición resultante es verdadera si todos los valores posibles de n hacen verdadera a $P(n)$. Falsa en caso contrario.
  - **Existencial**: $(\exists n) : (P(n))$
    - La proposición resultante es verdadera si al menos un valor posible de n hace verdadera a $P(n)$. Falsa en caso contrario.
- Alcance: El alcance de un cuantificador es la parte de la expresión en la cual el cuantificador ejerce su efecto.
  - Ejemplo: En la expresión $(\forall n) : (P(n) \lor Q(n))$, el alcance del cuantificador es toda la expresión $P(n) \lor Q(n)$.
  - En la expresión $(\exists n) : (P(n)) \land Q(n)$, el alcance del cuantificador es solo $P(n)$.

- Equivalencias:
  - $\neg(\forall n) : (P(n)) \Leftrightarrow (\exists n) : (\neg P(n))$
  - $\neg(\exists n) : (P(n)) \Leftrightarrow (\forall n) : (\neg P(n))$

Teoría de Conjuntos

Definición

- **Notación**:
  - $A = \{\dots\}$
  - $x \in A$ (x pertenece al conjunto A)
  - $x \notin A$ (x no pertenece al conjunto A)
    - Es equivalente a $\neg(x \in A)$
- **Conjunto**:
  - Colección de objetos bien definidos llamados elementos.
  - Se denotan comunmente con letras mayúsculas (A, B, C, etc) y se definen con llaves.
    - Ejemplo: $A = \{1, 2, 3\}$ es un conjunto con tres elementos: 1, 2 y 3.
  - Un conjunto puede ser finito o infinito.
- **Elemento**:
  - Cada elemento es único e "indivisible".
  - No hay ningún orden entre los elementos, es decir que por ejemplo: $A = \{1, 2, 3\}$ es equivalente a $B = \{3, 2, 1\}$, $A = B$.

Formas de expresar conjuntos

1. **Por extensión**:
   - Listando todos los elementos del conjunto.
   - Ejemplo: $A = \{a, b, c, d\}$
2. **Por comprensión**:
   - Describiendo una propiedad que caracteriza a los elementos del conjunto.
     - Ejemplo: $A = \{x \mid x \text{ es un número par y } 1 < x < 7\} = \{2, 4, 6\}$.
3. **Por diagramas de Venn**:
   - Representación gráfica de conjuntos y sus relaciones.

Conjunto vacío

- **Notación**: $\emptyset$ o $\{\}$.
- **Definición intuitiva**:
  - El conjunto vacío es el conjunto sin elementos.
  - Es decir, ningún objeto puede jamás pertenecer al conjunto vacío.
- **Definición formal**:
  - $(\forall x)(x \notin \emptyset)$

Operaciones

Subconjunto estricto

- **Notación**: $A \subset B$.
- **Definición intuitiva**:
  - A es un subconjunto estricto de B si todos los elementos de A están en B, pero B no es igual a A: porque B tiene uno o más elementos que no están en A.
- **Definición formal**:
  - $(\forall x)(x \in A \rightarrow x \in B) \land (\exists x)(x \in B \land x \notin A)$
- **Propiedades**:
  - Si A es subconjunto estricto de B, entonces A es subconjunto no estricto de B:
    - $A \subset B \implies A \subseteq B$
  - Ningún conjunto es subconjunto estricto de sí mismo:
    - $\forall A(A \not\subset A)$
- **Ejemplo**:
  - Si $A = \{1, 2\}$ y $B = \{1, 2, 3\}$ entonces $A \subset B$.

Subconjunto no estricto

- **Notación**: $A \subseteq B$.
- **Definición intuitiva**:
  - A es un subconjunto de B si todos los elementos de A están en B.
  - A puede ser igual a B.
- **Definición formal**:
  - $(\forall x)(x \in A \rightarrow x \in B)$
- **Propiedades**:
  - El conjunto vacío es subconjunto de cualquier conjunto:
    - $(\forall A)(\emptyset \subseteq A)$
  - Todo conjunto es subconjunto no estricto de sí mismo:
    - $(\forall A)(A \subseteq A)$

    - La inclusión es transitiva:
        - Si $A \subseteq B$ y $B \subseteq C$ entonces $A \subseteq C$.
- **Ejemplos**:
    - Si $A = \{1, 2\}$ y $B = \{1, 2, 3\}$ entonces $A \subseteq B$.
    - Si $A = \{1, 2, 3\}$ y $B = \{1, 2, 3\}$ entonces $A \subseteq B$.

Igualdad

- **Notación**: $A = B$.
- **Definición intuitiva**:
    - A es igual a B si ambos conjuntos tienen **exactamente** los mismos elementos.
- **Definición formal**:
    - $(\forall x)(x \in A \leftrightarrow x \in B)$
- **Equivalencia lógica con el subconjunto**:
    - $A = B \iff (A \subseteq B) \land (B \subseteq A)$
- **Ejemplo**:
    - Si $A = \{1, 2, 3\}$ y $B = \{1, 2, 3\}$ entonces $A = B$.
    - Si $A = \{1, 2, 3\}$ y $B = \{3, 2, 1\}$ entonces $A = B$.

Intersección

- **Notación**: $A \cap B$.
- **Definición intuitiva**:
    - La intersección de A y B es el conjunto de elementos que pertenecen a **ambos** conjuntos.
- **Definición formal**:
    - $A \cap B = \{x \mid x \in A \land x \in B\}$
- **Ejemplo**:
    - Si $A = \{1, 2, 3\}$ y $B = \{1, 2, 5\}$ entonces $A \cap B = \{1, 2\}$.

Unión

- **Notación**: $A \cup B$.
- **Definición intuitiva**:
    - La unión de A y B es el conjunto de elementos que pertenecen a A, a B, o a ambos.
    - Es decir, todos los elementos de A y todos los elementos de B.
    - Si hay repetidos entre A y B, se cuentan solo una vez.
- **Definición formal**:
    - $A \cup B = \{x \mid x \in A \lor x \in B\}$
- **Ejemplos**:
    - Si $A = \{1, 2, 3\}$ y $B = \{1, 2, 4\}$ entonces $A \cup B = \{1, 2, 3, 4\}$.
    - Si $A = \{1, 2, 3\}$ y $B = \{4, 5, 6\}$ entonces $A \cup B = \{1, 2, 3, 4, 5, 6\}$.

Diferencia

- **Notación**: $A - B$.
- **Definición intuitiva**:
  - La diferencia de A y B es el conjunto de elementos que pertenecen a A pero no pertenecen a B.
  - Algorítmicamente, para obtener $A - B$ se recorren todos los elementos de A uno por uno y se chequea si pertenecen a B. Si no pertenecen, se agregan al conjunto resultado.
- **Definición formal**:
  - $A - B = \{x \mid x \in A \land x \notin B\}$
- **Propiedades**:
  - Si $A \cap B = \emptyset$ entonces $A - B = A$.
  - $A - B \neq B - A$ en general.
- **Ejemplos**:
  - Si $A = \{1, 2, 3\}$ y $B = \{1, 2, 4\}$ entonces $A - B = \{3\}$.
  - Si $A = \{1, 2, 3\}$ y $B = \{4, 5, 6\}$ entonces $A - B = \{1, 2, 3\}$.

Complemento

- **Notación**: $\overline{A}$.
- **Definición intuitiva**:
  - El complemento de A es el conjunto de elementos que no pertenecen a A.
- **Definición formal**:
  - $\overline{A} = \{x \mid x \notin A\}$
- **Nota**: El complemento siempre se define respecto a un conjunto universal U, es decir, $\overline{A} = U - A$.
- **Ejemplo**:
  - Si el conjunto universal $U = \{1, 2, 3, 4, 5\}$ y $A = \{1, 2, 3\}$ entonces $\overline{A} = \{4, 5\}$.

Producto cartesiano

- **Notación**: $A \times B$.
- **Definición intuitiva**:
  - El producto cartesiano de A y B es el conjunto de todos los pares ordenados (x, y) donde x pertenece a A, y pertenece a B.
- **Definición formal**:
  - $A \times B = \{(x, y) \mid x \in A \land y \in B\}$
- **Nota 1**: El producto cartesiano se puede generalizar a más de dos conjuntos: $A_1 \times A_2 \times ... \times A_n = \{(x_1, x_2, ..., x_n) \mid x_1 \in A_1 \land x_2 \in A_2 \land ... \land x_n \in A_n\}$
- **Nota 2**: El producto cartesiano de un conjunto consigo mismo n veces se denota como $A^n = A \times A \times ... \times A$ (n veces).
- **Ejemplos**:
  - Si $A = \{1, 2\}$ y $B = \{3, 4\}$ entonces $A \times B = \{(1, 3), (1, 4), (2, 3), (2, 4)\}$.

- $\{1\}^5 = \{1\} \times \{1\} \times \{1\} \times \{1\} \times \{1\} = \{(1,1,1,1,1)\}$

Conjunto de partes

- **Notación**: $\mathcal{P}(A)$.
- **Definición intuitiva**:
  - El conjunto de partes de A es el conjunto de todos los subconjuntos de A, incluyendo el conjunto vacío y A mismo.
- **Definición formal**:
  - $\mathcal{P}(A) = \{B \mid B \subseteq A\}$
- **Nota 1**: Si A tiene n elementos, entonces $\mathcal{P}(A)$ tiene 2^n elementos.
- **Nota 2**: $(\forall A)\emptyset \in \mathcal{P}(A)$ y $A \in \mathcal{P}(A)$.
- **Nota 3**: Si $A \subseteq B$ entonces $\mathcal{P}(A) \subseteq \mathcal{P}(B)$.
- **Ejemplos**:
  - Si $A = \{1,2\}$, entonces $\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$.
  - $\mathcal{P}(\emptyset) = \{\emptyset\}$
  - $\mathcal{P}(\{\emptyset\}) = \{\emptyset, \{\emptyset\}\}$
  - $\mathcal{P}(\{\emptyset, \{\emptyset\}\}) = \{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\emptyset, \{\emptyset\}\}\}$

Equivalencias de las operaciones

1. **Conmutatividad**:
   i. $A \cup B = B \cup A$
   ii. $A \cap B = B \cap A$
2. **Asociatividad**:
   i. $A \cup (B \cup C) = (A \cup B) \cup C$
   ii. $A \cap (B \cap C) = (A \cap B) \cap C$
3. **Distributividad**:
   i. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
   ii. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
4. **Leyes de Morgan**:
   i. $\overline{A \cup B} = \overline{A} \cap \overline{B}$
   ii. $\overline{A \cap B} = \overline{A} \cup \overline{B}$
5. **Doble complemento**:
   i. $\overline{\overline{A}} = A$

Cardinalidad

Definición para conjuntos finitos

- Sea A un conjunto finito.
- La cardinalidad de A, denotada como $|A|$, es el número de elementos que tiene el conjunto A.
- Ejemplos:
  - Si $A = \{1, 2, 3, 4\}$, entonces $|A| = 4$.
  - Si $B = \{a, b, c\}$, entonces $|B| = 3$.
- Siempre se cumple que $|\mathcal{P}(A)| = 2^{|A|}$.
  - Ejemplo: Si $A = \{1, 2\}$, entonces $|\mathcal{P}(A)| = 2^2 = 4$. Esto es correcto ya que $\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$ tiene 4 elementos.
  - Se demuestra vía inducción.

Definición para conjuntos infinitos

Concepto

- La cardinalidad de un conjunto finito es un número, pero la cardinalidad de un conjunto infinito no lo es, sino que es una **propiedad** del conjunto llamada número cardinal. Éste nos permite hacer comparaciones de tamaño entre conjuntos infinitos.
- Para entender este concepto, se suele usar el principio del Palomar:
  - Si se tienen muchos objetos y varios cajones, pero menos cajones que objetos, entonces al menos un cajón debe contener más de un objeto.
  - Por ejemplo, si se tienen 10 objetos y 9 cajones, al menos un cajón debe contener más de un objeto.
  - La utilidad de este principio es que si un conjunto es más grande que otro, no se puede asignar cada elemento un lugar distinto en el más chico. En cierto punto, dos elementos tendrán que compartir lugar. Sirve para ver rápidamente que ciertas correspondencias entre conjuntos no pueden nunca existir.

Menor o igual

- A tiene cardinalidad menor o igual que B (denotado $|A| \leq |B|$) si y solo si se puede establecer una correspondencia uno a uno de cada elemento de A con un elemento distinto de B. Formalmente: $|A| \leq |B| \leftrightarrow (\exists f) : (\text{f es una función inyectiva tal que } f : A \to B)$.

Menor estricto

- A tiene cardinalidad menor que B (denotado $|A| < |B|$) si se cumplen simultáneamente $|A| \leq |B|$ y $|A| \neq |B|$. Formalmente: $|A| < |B| \leftrightarrow |A| \leq |B| \land |A| \neq |B|$. Esto es equivalente a decir que existe

una función inyectiva de A en B, pero no existe una función biyectiva entre A y B. Formalmente: $|A| < |B| \leftrightarrow (\exists f) : (\text{f es una función inyectiva tal que } f : A \to B) \wedge (\nexists g) : (\text{g es una función biyectiva tal que } g : A \to B)$.

Igualdad

- A tiene cardinalidad igual que B si se cumplen simultáneamente $|A| \leq |B|$ y $|B| \leq |A|$. Formalmente: $|A| = |B| \leftrightarrow |A| \leq |B| \wedge |B| \leq |A|$. Esto es equivalente a decir que existe una correspondencia biunívoca entre los elementos de A y los elementos de B. Formalmente: $|A| = |B| \leftrightarrow (\exists f) : (\text{f es una función biyectiva tal que } f : A \to B)$.

Utilidad de la función identidad

- Cuando se quiere demostrar que $|A| \leq |B|$ y se sabe que $A \subseteq B$, se puede usar la función identidad como función inyectiva de A en B. Lo que hace esta función es asignar a cada elemento de A el mismo elemento en B.

Ejemplos

- $\mathbb{N}$ (números naturales) y $\mathbb{N}^+$ (números naturales sin el cero) tienen la misma cardinalidad.
  i. Se demuestra que $|\mathbb{N}^+| \leq |\mathbb{N}|$ usando la función identidad: $f : \mathbb{N}^+ \to \mathbb{N}$ tal que $f(n) = n$.
  ii. Se demuestra que $|\mathbb{N}| \leq |\mathbb{N}^+|$ usando la función $g : \mathbb{N} \to \mathbb{N}^+$ tal que $g(n) = n + 1$.
  iii. Por lo tanto queda demostrado que $|\mathbb{N}| = |\mathbb{N}^+|$.
- $\mathbb{N}$ (números naturales) y P (números naturales pares) tienen la misma cardinalidad.
  i. Se demuestra que $|P| \leq |\mathbb{N}|$ usando la función identidad: $f : P \to \mathbb{N}$ tal que $f(n) = n$.
  ii. Se demuestra que $|\mathbb{N}| \leq |P|$ usando la función $g : \mathbb{N} \to P$ tal que $g(n) = 2n$.
  iii. Por lo tanto queda demostrado que $|\mathbb{N}| = |P|$.
- $\mathbb{N}$ (números naturales) y $\mathbb{N} \times \mathbb{N}$ (pares ordenados de números naturales) tienen la misma cardinalidad.
  i. Se demuestra que $|\mathbb{N} \times \mathbb{N}| \leq |\mathbb{N}|$ usando una función que genere un "ordenamiento" de los pares ordenados de números naturales. Por ejemplo, se puede usar la función $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ tal que $f(a, b) = \frac{(a+b)(a+b+1)}{2} + a$
  ii. Se demuestra que $|\mathbb{N}| \leq |\mathbb{N} \times \mathbb{N}|$ usando la función identidad $g : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ tal que $g(n) = (n, n)$.
  iii. Por lo tanto queda demostrado que $|\mathbb{N}| = |\mathbb{N} \times \mathbb{N}|$.

Conjuntos infinitos contables vs incontables

Contables

- Un conjunto infinito es **contable** si tiene la misma cardinalidad que $\mathbb{N}$ (números naturales).

- Para saber si un conjunto es contable alcanza con probar que $|A| \leq |\mathbb{N}|$ ya que si A es infinito se sabe que $|\mathbb{N}| \leq |A|$. Por lo tanto: Si $|A| \leq |\mathbb{N}| \implies \mathrm{A\ es\ contable}$
- **Propiedad**: La unión de dos conjuntos infinitos contables es un conjunto infinito contable.

Incontables

- Un conjunto infinito es **incontable** si no tiene la misma cardinalidad que $\mathbb{N}$ (números naturales).
- Para saber si un conjunto es incontable alcanza con probar que $|\mathbb{N}| < |A|$. Por lo tanto: Si $|\mathbb{N}| < |A| \implies \mathrm{A\ es\ incontable}$
- **El conjunto de partes de cualquier conjunto infinito contable es incontable**.
  - Por ejemplo, el conjunto de partes de los números naturales, $\mathcal{P}(\mathbb{N})$, es incontable, ya que $|\mathbb{N}| < |\mathcal{P}(\mathbb{N})|$.

Clasificación de cardinalidades de conjuntos infinitos típicos

- $\mathbb{N} = \mathbb{Q} = \mathbb{Z} < \mathbb{R} = \mathcal{P}(\mathbb{N})$
- Es decir, los números naturales, enteros y racionales son conjuntos infinitos **contables**, mientras que los números reales y el conjunto de partes de los números naturales son conjuntos infinitos **incontables**.
- Además existen muchos otros ejemplos, como que el conjunto de los numeros impares tiene la misma cardinalidad que los números naturales, y lo mismo con los números pares y los números naturales.

Máquinas de Turing (introducción)

Concepto

- Modelo matemático de un dispositivo de computación.
- Se compone de:
  - Una cinta infinita que se divide en infinitas celdas.
  - Un cabezal que se mueve sobre la cinta y que puede leer y escribir símbolos en cada celda.
- En cada instante de tiempo, la máquina está en un estado determinado de un conjunto finito de estados.
- La máquina puede cambiar de estado, escribir un símbolo en la celda actual y mover el cabezal a la izquierda o a la derecha, dependiendo del símbolo que está leyendo y del estado en el que se encuentra.
- Existe un símbolo especial llamado blanco y denotado con el símbolo "B", que representa una celda vacía.

Configuración inicial

- La máquina siempre inicia en el estado q_0.

- Si hay una cadena de entrada, esta se encuentra escrita en la cinta, comenzando desde la celda más a la izquierda. Además, esta cadena está delimitada por infinitos símbolos blancos (B) a la izquierda y a la derecha (principio y fin de la cadena en cuestión). No hay ningún blanco entre los símbolos de la cadena.
- Si no hay cadena de entrada, la cinta contiene solo símbolos blancos (B) en todas sus celdas.

Comportamiento

- Está definido por una función matemática de transición que indica, según el estado actual y el símbolo que está leyendo el cabezal:
  - El nuevo estado al que la máquina transicionará.
  - El símbolo que escribirá en la celda actual (puede ser el mismo que estaba leyendo).
  - La dirección en la que moverá el cabezal (izquierda o derecha).
- Si la función de transición no está definida para el estado actual y el símbolo que está leyendo, la máquina se detiene.
- **El resultado del cómputo de la MT es la cadena que queda escrita en la cinta cuando la máquina se detiene**.

MT General

- **Notación**: $M = \langle Q, \Sigma, \Gamma, \delta, q_0 \rangle$
  - Q: Conjunto finito de estados.
  - Σ: Alfabeto de la entrada.
  - Γ: Alfabeto de la cinta $\left(\Sigma \subset \Gamma, B \in (\Gamma - \Sigma)\right)$.
  - δ: Función de transición ($\delta : Q \times \Gamma \to Q \times \Gamma \times \{D, I\}$).
  - q_0: Estado inicial ($q_0 \in Q$).
- **Nota 1**: $Q \cap \Gamma = \emptyset$, es decir, ningun estado puede tener el mismo nombre que un símbolo del alfabeto de la cinta.
- **Nota 2:** Σ nunca puede ser igual a Γ, ya que Γ siempre tiene el símbolo de blanco B pero Σ nunca lo tiene.

MT Alternativa

- **Notación**: $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_d \rangle$
  - Q: Conjunto finito de estados.
  - Σ: Alfabeto de la entrada.
  - Γ: Alfabeto de la cinta $\left(\Sigma \subset \Gamma, B \in (\Gamma - \Sigma)\right)$.
  - δ: Función de transición ($\delta : Q \times \Gamma \to Q \cup \{q_d\} \times \Gamma \times \{D, I\}$).
  - q_0: Estado inicial ($q_0 \in Q$).
  - q_d: Estado de detención ($q_d \notin Q$).
- La máquina deja de computar cuando llega a q_d porque q_d no es parte del dominio de δ.

MT Reconocedora

- **Notación**: $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$
  - Q: Conjunto finito de estados.
  - Σ: Alfabeto de la entrada.
  - Γ: Alfabeto de la cinta ($\Sigma \subset \Gamma$).
  - δ: Función de transición ($\delta : Q \times \Gamma \to Q \cup \{q_A, q_R\} \times \Gamma \times \{D, I\}$).
  - q_0: Estado inicial ($q_0 \in Q$).
  - q_A: Estado de aceptación ($q_A \notin Q$).
  - q_R: Estado de rechazo ($q_R \notin Q$).
- **Estados de detención**:
  - q_A (Aceptación): Si la máquina llega a este estado, se dice que acepta la cadena de entrada.
  - q_R (Rechazo): Si la máquina llega a este estado, se dice que rechaza la cadena de entrada.
  - En ambos casos, la máquina se detiene. Si no llega nunca a ninguno de estos estados, entonces nunca se detendrá.
- **Lenguaje que acepta o reconoce**:
  - El lenguaje que acepta o reconoce una máquina de Turing reconocedora M es el conjunto de todas las cadenas que M **acepta**.
  - Formalmente: $L(M) = \{w \in \Sigma^* \mid M \text{ acepta } w\}$
  - Dicho de otra forma:
    - $w \in L(M) \leftrightarrow$ con entrada w, M termina en el estado q_A.
    - Para $v \notin L(M)$, se tiene que M termina en el estado q_R o M loopea infinitamente.

Lenguajes Formales

Símbolo

- **Notación**: $x, y, z, \ldots$
- **Definición intuitiva**:
  - Un símbolo es un objeto indivisible.
- **Definición formal**:
  - x es un símbolo si y solo si:
    - x es un objeto indivisible.
  - Un símbolo es un objeto indivisible que pertenece a un conjunto **finito** llamado **alfabeto**.
- **Ejemplo**:
  - a, b, c, 1, 2, 3, @, #, $, %, &, ...

Alfabeto

- **Notación**: Σ
- **Definición intuitiva**:
  - Un alfabeto es un conjunto **no vacío** y **finito** de símbolos.
- **Definición formal**:
  - Σ es un alfabeto si y solo si:
    - $\Sigma \neq \emptyset$
    - Σ es finito
    - $(\forall x)(x \in \Sigma \rightarrow \text{x es un símbolo indivisible})$
- **Ejemplo**:
  - $\Sigma = \{a, b, c\}$ es un alfabeto con tres símbolos: a, b y c. ✅
  - $\Sigma = \{aa, b, c\}$ no es un alfabeto porque "aa" no es un símbolo indivisible. ❌

Cadena / palabra / sentencia / string

- **Notación**: w o λ (para la cadena vacía)
- **Definición intuitiva**:
  - Una cadena es una secuencia **finita** de símbolos tomados de un **alfabeto** Σ.
- **Definición formal**:
  - w es una cadena si y solo si:
    - $w = x_1x_2...x_n$, donde $n \geq 0$ y $x_i \in \Sigma$ para $1 \leq i \leq n$.
    - Si $n = 0$, entonces $w = \lambda$ (cadena vacía, sin ningún símbolo).
- **Ejemplo**:
  - Si $\Sigma = \{a, b\}$, entonces algunas cadenas posibles son:
    - $w_1 = abba$
    - $w_2 = aab$
    - $w_3 = b$
    - $w_4 = \lambda$ (cadena vacía)
- **Longitud de una cadena**:
  - Se denota como $|w|$ y es el número de símbolos en la cadena w.
  - Ejemplo: Si $w = abba$, entonces $|w| = 4$.
  - Para la cadena vacía λ, $|\lambda| = 0$.
- **Concatenación de cadenas**:
  - Si w_1 y w_2 son cadenas, la concatenación de w_1 y w_2 se denota como w_1w_2 y es la cadena formada al unir w_1 seguido de w_2.
    - Ejemplo: Si $w_1 = ca$ y $w_2 = sa$, entonces $w_1w_2 = casa$.
  - Esta operación es **asociativa** pero no **conmutativa**: $(w_1w_2)w_3 = w_1(w_2w_3)$ pero $w_1w_2 \neq w_2w_1$ en general.

- La cadena vacía λ actúa como el elemento neutro en la concatenación: $w\lambda = \lambda w = w$ para cualquier cadena w.
- La concatenación de cadenas puede aumentar la longitud: $|w_1 w_2| = |w_1| + |w_2|$.
  - Ejemplo: Si $w_1 = ab$ y $w_2 = ba$, entonces $w_1 w_2 = abba$ y $|w_1 w_2| = 4$.

- **Potencia i-ésima de una cadena**:
  - La potencia i-ésima de una cadena w, denotada como w^i, es la concatenación de i copias de w.
    - Si $i = 0$, entonces $w^0 = \lambda$ (cadena vacía).
    - Si $i > 0$, entonces $w^{(i+1)} = ww^i (\forall i \geq 0)$.
    - Ejemplo: Si $w = ab$ y $i = 3$, entonces $w^3 = ababab$.
  - Por convención, $w^0 = \lambda$ para cualquier cadena w.
- **Inverso de una cadena**:
  - El inverso de una cadena $w = x_1 x_2 ... x_n$ se denota como w^R y es la cadena formada al escribir los símbolos de w en orden inverso: $w^R = x_n x_{n-1} ... x_1$.
    - Ejemplo: Si $w = auto$, entonces $w^R = otua$.

Sigma estrella

- **Notación**: Σ^*
- **Definición intuitiva**:
  - Σ^* es el conjunto de todas las cadenas (incluyendo la cadena vacía) que se pueden formar con los símbolos del alfabeto Σ.
  - Es un conjunto infinito contable.
- **Definición formal**:
  - $\Sigma^* = \{w \mid w \text{ es una cadena (incluyendo la cadena vacía) formada por símbolos de } \Sigma\}$
- **Ejemplo**:
  - Si $\Sigma = \{a, b\}$, entonces:
    - $\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, ...\}$

Lenguaje

- **Notación**: L
- **Definición intuitiva**:
  - Un lenguaje es un **conjunto** de una o más cadenas formadas a partir de un alfabeto Σ.
- **Definición formal**:
  - L es un lenguaje si y solo si $L \subseteq \Sigma^*$
- **Ejemplo**:
  - Si $\Sigma = \{0, 1\}$, entonces:
    - $\Sigma^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, ...\}$
  - Posibles lenguajes definidos sobre Σ podrían ser:

        - $L_1 = \{0, 1, 00, 01\}$
        - $L_2 = \emptyset$
        - $L_3 = \{\lambda\}$
        - $L_4 = \Sigma^*$
        - $L_5 = \{1w \mid w \in \Sigma^*\}$
    - Los siguientes NO son lenguajes sobre Σ:
        - $L_6 = \{2, 3\}$ (los símbolos 2 y 3 no pertenecen al alfabeto Σ)
        - $L_7 = 5$ (5 no es un conjunto de cadenas, es un número)
- **Nota 1**: Si L es un lenguaje sobre el alfabeto Σ, su complemento también es un lenguaje sobre el mismo alfabeto, definido como $\overline{L} = \Sigma^* - L$.
- **Nota 2**: Por definición, para cualquier lenguaje L se cumple:
    - $L \cup \overline{L} = \Sigma^*$
    - $L \cap \overline{L} = \emptyset$
- **L cursiva ($\mathscr{L}$)**:
    - $\mathscr{L}$ es el conjunto de todos los lenguajes posibles sobre un alfabeto Σ.
    - Es decir, $\mathscr{L} = \mathcal{P}(\Sigma^*)$.

Máquinas de Turing (continuación)

Definición de equivalencia de MT

- **Definición intuitiva**:
    - Una máquina de Turing M_1 es equivalente a una máquina de Turing M_2 si y solo si ambas máquinas reconocen el mismo lenguaje.
- **Definición formal**:
    - $M_1 \equiv M_2 \leftrightarrow L(M_1) = L(M_2)$

Definición de equivalencia de modelo de MT

- **Definición intuitiva**:
    - Dos modelos de máquinas de Turing son equivalentes si y solo si para cada máquina de un modelo, existe una máquina en el otro modelo que reconoce el mismo lenguaje.
    - Esto implica que ambos modelos tienen la misma capacidad de reconocimiento de lenguajes.
- **Definición formal**:
    - Dos modelos de máquinas de Turing A y B son equivalentes si y solo si:
        - $(\forall M_A \in A)(\exists M_B \in B)(M_A \equiv M_B)$ y

- $(\forall M_B \in B)(\exists M_A \in A)(M_B \equiv M_A)$

Modelo D-I-S

- Es una máquina de Turing que admite transiciones sin mover el cabezal.
- **Notación**: $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$
  - Q: Conjunto finito de estados.
  - Σ: Alfabeto de la entrada.
  - Γ: Alfabeto de la cinta ($\Sigma \subseteq \Gamma$).
  - δ: Función de transición ($\delta : Q \times \Gamma \to Q \cup \{q_A, q_R\} \times \Gamma \times \{D, I, S\}$).
  - q_0: Estado inicial ($q_0 \in Q$).
  - q_A: Estado de aceptación ($q_A \notin Q$).
  - q_R: Estado de rechazo ($q_R \notin Q$).

Modelo k-cintas

- Es una máquina de Turing que tiene k cintas y k cabezales que pueden moverse de forma independiente.
- El input se encuentra en la primer cinta y las demás cintas están inicialmente vacías.
- **Notación**: $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$
  - Q: Conjunto finito de estados.
  - Σ: Alfabeto de la entrada.
  - Γ: Alfabeto de la cinta ($\Sigma \subseteq \Gamma$).
  - δ: Función de transición ($\delta : Q \times \Gamma^k \to Q \cup \{q_A, q_R\} \times (\Gamma \times \{D, I, S\})^k$).
  - q_0: Estado inicial ($q_0 \in Q$).
  - q_A: Estado de aceptación ($q_A \notin Q$).
  - q_R: Estado de rechazo ($q_R \notin Q$).

Clasificación de lenguajes

Lenguajes recursivamente enumerables (RE)

- **Definición intuitiva**:
  - Un lenguaje es recursivamente enumerable ($L \in RE$) si existe una máquina de Turing reconocedora que acepta todas las cadenas del lenguaje, pero **puede no detenerse para las cadenas que no pertenecen al lenguaje**.
- **Definición formal**:

- $L \in RE \leftrightarrow \exists M \mid L(M) = L$.
- Si $w \in L$ entonces:
  - M **siempre se detiene** en el estado de aceptación q_A.
- Si $w \notin L$ entonces:
  - M se detiene en el estado de rechazo q_R o **no se detiene y loopea infinitamente**. No es posible saber cuál de los dos casos ocurrirá de antemano.

- **Nota**: Todo lenguaje FINITO es recursivamente enumerable.

Lenguajes recursivos o decidibles (R)

- **Definición intuitiva**:
  - Un lenguaje es recursivo o decidible ($L \in R$) si existe una máquina de Turing reconocedora que acepta todas las cadenas del lenguaje y **se detiene para todas las cadenas, ya sea aceptándolas o rechazándolas**.
- **Definición formal**:
  - $L \in R \leftrightarrow \exists M \mid L(M) = L$ y M $\text{se detiene para todas las cadenas}$.
  - Si $w \in L$ entonces:
    - M **siempre se detiene** en el estado de aceptación q_A.
  - Si $w \notin L$ entonces:
    - M **siempre se detiene** en el estado de rechazo q_R.
- **Nota 1**: Todo lenguaje FINITO es decidible.
- **Nota 2**: La intersección y la unión de dos lenguajes decidibles da como resultado otro lenguaje decidible.
  - Si $L_1, L_2 \in R$ entonces $L_1 \cap L_2 \in R$ y $L_1 \cup L_2 \in R$.

Lenguajes CO-R

- **Definición intuitiva**:
  - Un lenguaje es CO-R ($L \in \text{CO-R}$) si y solo si su complemento respecto de Σ^* es decidible.
- **Definición formal**:
  - $L \in \text{CO-R} \leftrightarrow \overline{L} \in R$
  - $\overline{L} = \Sigma^* - L$

Lenguajes CO-RE

- **Definición intuitiva**:
  - Un lenguaje es CO-RE ($L \in \text{CO-RE}$) si y solo si su complemento respecto de Σ^* es recursivamente enumerable.
- **Definición formal**:
  - $L \in \text{CO-RE} \leftrightarrow \overline{L} \in RE$

- $\overline{L} = \Sigma^* - L$

Teoremas

1. **Todo lenguaje decidible es recursivamente enumerable, pero no todo lenguaje recursivamente enumerable es decidible**: $R \subseteq RE$
2. **Todo lenguaje decidible es CO-R**: $R \subseteq \text{CO-R}$
3. **Todo lenguaje CO-R es decidible**: $\text{CO-R} \subseteq R$
4. **Por lo tanto, para cualquier lenguaje decidible, su complemento también es decidible, y además todo lenguaje CO-R es recursivamente enumerable**: $R = \text{CO-R}$ y $\text{CO-R} \subseteq RE$
5. **Todo lenguaje recursivo es CO-RE**: $R \subseteq \text{CO-RE}$
6. **Por lo tanto, todo lenguaje decidible es CO-RE y viceversa, y además todo lenguaje CO-RE es recursivamente enumerable**: $R = \text{CO-RE}$ y $R \subseteq (RE \cap \text{CO-RE})$
7. **Los lenguajes que son tanto RE como CO-RE son decidibles**: $(RE \cap \text{CO-RE}) \subseteq R$
8. **Los lenguajes decidibles son exactamente los mismos que los que son RE y CO-RE a la vez**: $R = (RE \cap \text{CO-RE})$

Orden canónico para Σ^*

- Sea Σ un alfabeto finito.
- El orden canónico para Σ^* es una forma de listar todas las cadenas posibles formadas con los símbolos de Σ en orden lexicográfico.
- Ejemplo:
  - Si $\Sigma = \{0, 1\}$, entonces el orden canónico para Σ^* es:
    - $\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, ...$
    - Si w es un string de $\{0, 1\}^*$, la posición i que ocupa en el orden canónico se escribe en binario como $1w$. Decimos entonces que w es el i-ésimo string y por ello lo denotamos w_i.
    - Por ejemplo el string λ ocupa la posición 1 (1λ) el string 01 ocupa la posición 5 (101), el string 100 la posición 12 (1100), etc.

Codificación binaria de una MT

- Es una forma de representar una máquina de Turing como una cadena binaria para que la misma pueda ser usada como entrada de otra máquina de Turing.
- Ejemplo:
  - Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$ una máquina de Turing reconocedora D-I-S tal que:
  - $Q = \{q_0\}$
  - $\Sigma = \{a, b\}$
  - $\Gamma = \{B, a, b\}$

    - $\delta(q_0, a) = (q_A, a, D)$
    - $\delta(q_0, b) = (q_R, b, S)$
    - $\delta(q_0, B) = (q_R, B, S)$
    - Donde se codifican los símbolos de la siguiente manera:
        - $q_0 = 111$
        - $q_R = 11$
        - $q_A = 1$
        - $b = 111$
        - $a = 11$
        - $B = 1$
        - $S = 111$
        - $I = 11$
        - $D = 1$
    - Entonces la codificación binaria de M es (usamos dos ceros para separar las funciones de transición):
        - $111_{q_0}011_a01_{q_a}011_a01_D00_{sig}111_{q_0}0111_b011_{q_R}0111_b0111_S00_{sig}111_{q_0}01_B011_{q_R}01_B0111_S$
- Hay N! formas de codificar una MT, si tenemos N funciones de transición. En este caso 3! = 6.
- Se denomina i-ésima máquina de Turing y se denota M_i a la MT cuyo código binario es el i-ésimo string en el orden canónico de $\{0,1\}^*$, es decir $\langle M_i \rangle = w_i$.
    - Si se da el caso que w_i no es un código válido de una MT, entonces M_i es una MT ficticia que rechaza todo, es decir $L(M_i) = \emptyset$.
    - Por lo tanto $\forall w_i \in \{0,1\}^* \exists M_i$.

Lenguaje diagonal $\left(L_D\right)$

- **Definición intuitiva**:
    - El lenguaje diagonal L_D es el conjunto de todas las cadenas que no son aceptadas por su **correspondiente** máquina de Turing. Consiste en las codificaciones de MT que rechazan su propia codificación.
- **Definición formal**:
    - $\Sigma = \{0,1\}$
    - $w_i = \text{i-ésimo string en orden canónico de } \Sigma^*$
    - $M_i = \text{i-ésima máquina de Turing}$
    - $L_D = \{w_i \in \Sigma^* \mid w_i \notin L(M_i)\}$
- Este lenguaje no es recursivamente enumerable, por lo tanto no es decidible tampoco y no existe una máquina de Turing que lo reconozca.
    - Es decir, no es posible construir una MT que diga, dada una cadena w_i que representa una codificación de una MT M_i, si M_i acepta o rechaza su propia codificación w_i.
- **Ejemplo**:
    - w_{10} es el décimo string en orden canónico de $\{0,1\}^*$, es decir $w_{10} = 1010$.

    - M_{10} es la codificación de la máquina de Turing correspondiente a w_{10}, es decir $\langle M \rangle = 1010$. Obviamente esta codificación no corresponde a una MT válida, porque no hay suficientes bits para codificar todas sus partes. Pero si asumimos que M_{10} es una MT ficticia que rechaza todo, es decir $L(M_{10}) = \emptyset$, entonces $w_{10} \in L_D$ porque $w_{10} \notin L(M_{10})$.
- **Teoremas**:
    - $L_D \notin RE$ y por lo tanto obviamente $L_D \notin R$
    - $L_D \in \text{CO-RE}$
    - $\overline{L_D} \in RE$
    - $\overline{L_D} \in (RE - R)$

Máquina de Turing Universal

- Es una máquina de Turing que recibe como entrada la codificación binaria de otra máquina de Turing junto con una cadena de entrada para esa máquina, y simula el comportamiento de esa máquina con esa cadena de entrada, ejecutándola.

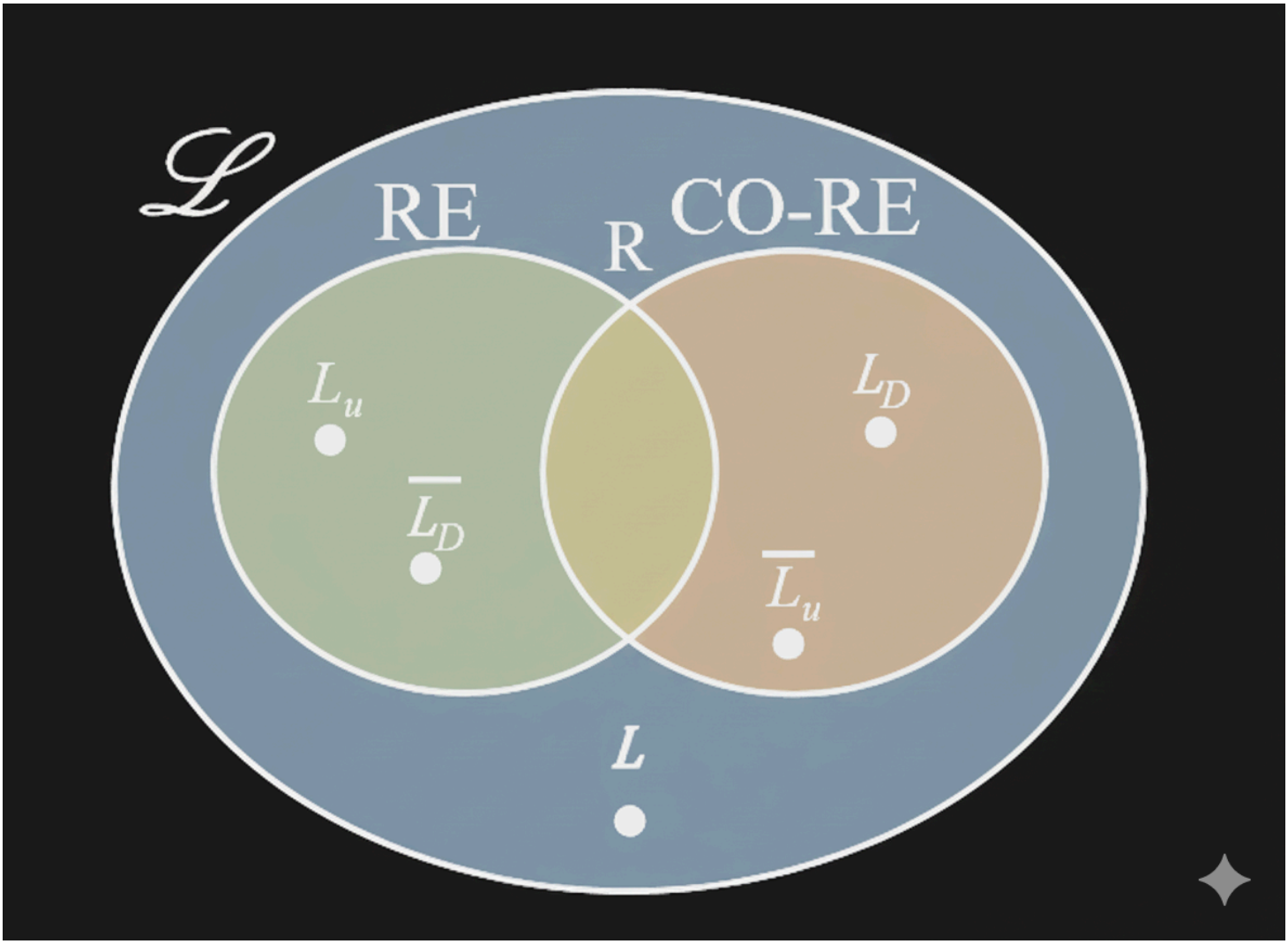
Lenguaje Universal L_u

- **Definición intuitiva**: El lenguaje universal L_u es el conjunto de todos los pares (máquina, cadena) tales que la máquina acepta esa cadena.
- **Definición formal**:
    - $L_u = \{(\langle M \rangle, w) \mid w \in L(M)\}$
    - $L_u = L(M_u)$
- **Teoremas**:
    - $L_u \in RE$
    - $L_u \notin R$
    - Por lo tanto: $L_u \in (RE - R)$
    - $\overline{L_u} \notin RE$

Lenguaje L

- **Definición intuitiva**:
    - Se trata de un lenguaje que no es recursivamente enumerable (y por lo tanto tampoco decidible) y además no es CO-RE.
- **Definición formal**:
    - $L = \{1w \mid w \in L_D\} \cup \{0w \mid w \notin L_D\}$
- **Teorema**: $L \in (\mathscr{L} - (RE \cup \text{CO-RE}))$

Diagrama de Venn de los lenguajes vistos



Reducibilidad

Definición de reducción

- **Definición intuitiva**:
  - Una reducibilidad es una función que transforma instancias de un problema/lenguaje en otro.
  - A cada palabra de L_1 se le asigna una palabra de L_2.
  - A cada palabra que NO pertenece a L_1 se le asigna una palabra que NO pertenece a L_2.
- **Definición formal**:
  - Sean L_1 y L_2 dos lenguajes sobre un alfabeto Σ.
  - L_1 se reduce a L_2 (denotado $L_1 \ \alpha \ L_2$) si existe una **función total computable** $f : \Sigma^* \to \Sigma^*$ tal que:

$$(\forall w \in \Sigma^*)(w \in L_1 \leftrightarrow f(w) \in L_2)$$

- Equivalentemente:

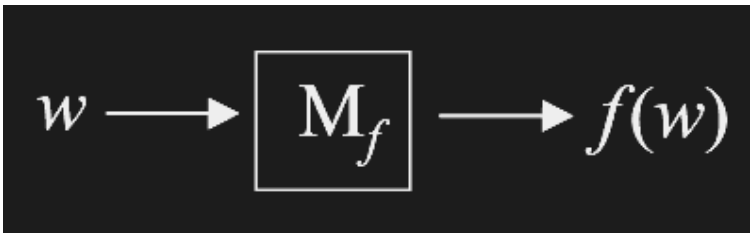
$$(\forall w \in \Sigma^*)((w \in L_1 \rightarrow f(w) \in L_2) \land (f(w) \in L_2 \rightarrow w \in L_1))$$

- O, por contrarecíproco $(A \rightarrow B \Leftrightarrow \neg B \rightarrow \neg A)$ de lo anterior:

$$(\forall w \in \Sigma^*)((w \in L_1 \rightarrow f(w) \in L_2) \land (w \notin L_1 \rightarrow f(w) \notin L_2)$$

Función total computable

- Una función f es **computable** si existe una máquina de Turing que la computa y que siempre se detiene.
- Una función f es **total** si está definida para todos los elementos de su dominio, es decir, se debe poder aplicar a cualquier cadena de Σ^*.



- M_f nunca loopea. La expresión $M_f(w)$ se refiere a la función computada por la máquina de Turing M_f.

Ejemplo 1

- Sea $L_1 = \{w \in \{a,b\}^* \mid w \text{ comienza con a}\}$
- Sea $L_2 = \{w \in \{a,b\}^* \mid w \text{ comienza con b}\}$

Se demostrará que existe una reducción: $L_1 \ \alpha \ L_2$.

Para esto, queremos encontrar una función total computable $f : \{a,b\}^* \rightarrow \{a,b\}^*$ tal que:

$$(\forall w \in \{a,b\}^*)(w \in L_1 \leftrightarrow f(w) \in L_2)$$

Para lo anterior, necesitamos encontrar una MT M_f que compute la función f mencionada y que siempre se detenga.

M_f es una MT de cómputo y se define de la siguiente manera:

$M_f = \langle Q, \Sigma, \Gamma, \delta, q_0, q_d \rangle$ donde:

- $Q = \{q_0\}$
- $\Sigma = \{a,b\}$
- $\Gamma = \{a,b,B\}$

Y su función de transición δ es:

- $\delta(q_0, a) = (q_d, b, S)$
- $\delta(q_0, b) = (q_d, a, S)$
- $\delta(q_0, B) = (q_d, B, S)$

Es decir, M_f lee **únicamente el primer símbolo de la cadena de entrada** y lo reemplaza por el símbolo "opuesto" (a por b, b por a). Si la cadena está vacía, no hace nada.

Podemos ver que:

1. M_f **siempre se detiene**, ya que en cualquier caso llega al estado de detención q_d después de leer el primer símbolo.
2. f es **total**, ya que está definida para todas las cadenas en $\{a, b\}^*$.
3. La función f cumple con la **condición de reducción**:
   - $(\forall w \in \Sigma^*)(w \in L_1 \rightarrow f(w) \in L_2)$ se cumple porque si w comienza con "a", entonces $f(w)$ comenzará con "b", por lo cual se cumple que $f(w) \in L_2$, ya que cumple la condición de L_2.
   - $(\forall w \in \Sigma^*)(w \notin L_1 \rightarrow f(w) \notin L_2)$ se cumple porque si w no comienza con "a" (es decir, comienza con "b" o es la cadena vacía), entonces $f(w)$ no comenzará con "b" (comenzará con "a" o será vacío).

Ejemplo 2

- Sea $L_1 = \{w \in \{0,1\}^* \mid \text{Cantidad de unos de w es par}\}$
- Sea $L_2 = \{w \in \{0,1\}^* \mid \text{Cantidad de unos de w es impar}\}$

Se demostrará que existe una reducción: $L_1 \ \alpha \ L_2$.

Para esto, queremos encontrar una función total computable $f : \{0,1\}^* \rightarrow \{0,1\}^*$ tal que:

$$(\forall w \in \{0,1\}^*)(w \in L_1 \leftrightarrow f(w) \in L_2)$$

Para lo anterior, necesitamos encontrar una MT M_f que compute la función f mencionada y que siempre se detenga.

M_f es una MT de cómputo y se define de la siguiente manera:

$M_f = \langle Q, \Sigma, \Gamma, \delta, q_0, q_d \rangle$ donde:

- $Q = \{q_0, q_1\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, B\}$

Y su función de transición δ es:

- $\delta(q_0, 0) = (q_1, 0, I)$
- $\delta(q_0, 1) = (q_1, 1, I)$
- $\delta(q_0, B) = (q_1, B, I)$

- $\delta(q_1, B) = (q_d, 1, S)$

Es decir, M_f se mueve una posición a la izquierda al comenzar, y como está al inicio de la cadena, siempre llega a un símbolo blanco. En ese momento escribe un 1 en ese lugar y se detiene.

Podemos ver que:

1. M_f **siempre se detiene**, ya que en cualquier caso llega al estado de detención q_d después de leer el primer símbolo y moverse a la izquierda, dado que siempre hay blanco a la izquierda del inicio de la cadena.
2. f es **total**, ya que está definida para todas las cadenas en $\{0, 1\}^*$.
3. La función f cumple con la **condición de reducción**:
   - $(\forall w \in \Sigma^*)(w \in L_1 \rightarrow f(w) \in L_2)$ se cumple porque si w tiene una cantidad par de unos, al agregarle un 1 al inicio, la cantidad de unos en $f(w)$ siempre será impar, por lo cual se cumple que $f(w) \in L_2$, ya que cumple la condición de L_2 que es tener una cantidad impar de unos.
   - $(\forall w \in \Sigma^*)(w \notin L_1 \rightarrow f(w) \notin L_2)$ se cumple porque si w tiene una cantidad impar de unos, al agregarle un 1 al inicio, la cantidad de unos en $f(w)$ siempre será par, por lo cual se cumple que $f(w) \notin L_2$, ya que no cumple la condición de L_2.
   - **En ambos casos se aprovecha la propiedad matemática**:

$$\text{par} + 1 = \text{impar}$$

$$\text{impar} + 1 = \text{par}$$

Implicaciones de la reducción

Sean L_1 y L_2 dos lenguajes. $L_1 \ \alpha \ L_2$ implica que:

1. Se puede construir una MT que acepte L_1 a partir de la MT que acepta L_2 (si existe).
2. Intuitivamente, L_1 no puede ser más difícil computacionalmente que L_2, porque se puede usar L_2 para resolver L_1.
3. Intuitivamente, la reducción establece una relación de $\leq$ grado de dificultad computacional entre los lenguajes. Si $L_1 \ \alpha \ L_2$, entonces L_1 no es más difícil que L_2, es o igual de difícil o más fácil. Si puedo computar L_2, entonces puedo computar L_1.

Teoremas fundamentales

Sobre R

- Sean L_1 y L_2 dos lenguajes tal que $L_1 \ \alpha \ L_2$. Entonces:
  - $L_2 \in R \rightarrow L_1 \in R$
    - Si L_2 es decidible, entonces L_1 también.
  - $L_1 \notin R \rightarrow L_2 \notin R$ (por contrarrecíproco)

        - Si L_1 NO es decidible, entonces L_2 tampoco.
- Este teorema nos sirve para poder determinar si un lenguaje es decidible o no, partiendo de saber que otro lenguaje al que se reduce es decidible o no.

Sobre RE

- Sean L_1 y L_2 dos lenguajes tal que $L_1 \ \alpha \ L_2$. Entonces:
    - $L_2 \in RE \rightarrow L_1 \in RE$
        - Si L_2 es recursivamente enumerable, entonces L_1 también.
    - $L_1 \notin RE \rightarrow L_2 \notin RE$ (por contrarrecíproco)
        - Si L_1 NO es recursivamente enumerable, entonces L_2 tampoco.
- Este teorema nos sirve para poder determinar si un lenguaje es recursivamente enumerable o no, partiendo de saber que otro lenguaje al que se reduce es recursivamente enumerable o no.

Halting Problem (HP)

Definición

- **Definición intuitiva**:
    - Este lenguaje contiene las codificaciones de todas las máquinas de Turing que se detienen al ejecutarse con una cadena de entrada w, ya sea aceptando o rechazando.
- **Definición formal**:
    - El lenguaje Halting Problem se define como:

$$HP = \{(\langle M \rangle, w) \mid \text{M es una MT que se detiene con input w}\}$$

- Este lenguaje es **recursivamente enumerable pero no decidible**, es decir:

$$HP \in (RE - R) \Leftrightarrow (HP \in RE) \wedge (HP \notin R)$$

Lenguaje L_{Σ^*}

Definición

- **Definición intuitiva**:
    - Este lenguaje contiene las codificaciones de todas las máquinas de Turing que aceptan todas las cadenas posibles sobre el alfabeto Σ.

- **Definición formal**:
  - El lenguaje L_{Σ^*} se define como:

$$L_{\Sigma^*} = \{\langle M \rangle \mid \text{M es una MT y } L(M) = \Sigma^*\}$$

- Este lenguaje no es recursivamente enumerable y por lo tanto tampoco decidible:
  - $L_{\Sigma^*} \notin RE$
  - $L_{\Sigma^*} \notin R$

Lenguaje L_{EQ}

Definición

- **Definición intuitiva**:
  - Este lenguaje contiene los pares de codificaciones de máquinas de Turing que reconocen el mismo lenguaje.
- **Definición formal**:
  - El lenguaje L_{EQ} se define como:

$$L_{EQ} = \{(\langle M_1 \rangle, \langle M_2 \rangle) \mid L(M_1) = L(M_2)\}$$

- Este lenguaje no es recursivamente enumerable y por lo tanto tampoco decidible:
  - $L_{EQ} \notin RE$
  - $L_{EQ} \notin R$

Notación Asintótica

Contexto

- La notación asintótica está asociada al crecimiento de las funciones matemáticas.
- En el contexto de la computación, asociamos estas funciones a la cantidad de operaciones o tareas que el algoritmo debe realizar en función del tamaño de la entrada.
- Normalmente, a mayor entrada, mayor cantidad de operaciones (monotonía).
- La notación asintótica nos permite describir el comportamiento de estas funciones cuando la entrada crece arbitrariamente, es decir, tiende a infinito.

Notaciones útiles

- **Conjuntos**:
  - $\mathbb{N}$ es el conjunto de todos los números naturales, **incluyendo el cero**.
  - $\mathbb{R}^{\geq 0}$ es el conjunto de todos los números reales positivos, **incluyendo el cero**.
  - $\mathbb{R}^{+}$ es el conjunto de todos los números reales positivos, **excluyendo el cero**.
- **Cuantificadores**:
  - $\exists$: Existe al menos un elemento que satisface la propiedad.
  - $\exists^{\infty}$: Existen **infinitos** elementos que satisfacen la propiedad.
  - $\forall$: Todos los elementos satisfacen la propiedad, sin excepción.
  - $\forall^{\infty}$: Todos los elementos excepto un número **finito** de ellos satisfacen la propiedad.

Orden O

Primera definición

- Sea $t(n)$ una función.
- Decimos que $t(n)$ está en el orden de $f(n)$ ($t(n) \in O(f(n))$) si:
  - $\exists c \in \mathbb{R}, c > 0$
  - $\exists u_0 \in \mathbb{N}, u_0 > 0$
  - Tal que $t(n) \leq c \cdot f(n) \quad \forall n > u_0$
- Intuitivamente, $t(n) \in O(f(n))$ significa que la función $t(n)$ crece **a lo sumo tan rápido como** $f(n)$ cuando n tiende a infinito. Puede crecer más lento, pero no más rápido.

Segunda definición

- $O(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid (\exists c \in \mathbb{R}^{+})(\forall^{\infty} n \in \mathbb{N})\ [t(n) \leq c \cdot f(n)]\}$
  - Es decir, es un conjunto de funciones de los naturales a los reales mayores o iguales a cero, tales que existe una constante real positiva y hay infinitos naturales para los cuales se cumple que la función $t(n)$ es acotada superiormente por $c \cdot f(n)$.

Tercera definición

- $O(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^{+} \mid \exists c \in \mathbb{R}^{+},\ n_0 \in \mathbb{N} \mid t(n) \leq c \cdot f(n), n \geq n_0\}$
  - Esta es una definición alternativa, donde en vez de ir de los naturales a los reales mayores o iguales a cero, va de los naturales a los reales estrictamente mayores que cero, y además se reemplaza el cuantificador $\forall^{\infty}$ por el umbral, es decir, existe un n_0 a partir del cual se cumple la condición.
  - Es una cota **superior**.

Orden Omega (Ω)

- La definición es muy similar a la de Orden O, pero en este caso se trata de una cota **inferior**:
- $\Omega(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+,\ n_0 \in \mathbb{N} \mid t(n) \geq c \cdot f(n), n \geq n_0\}$
- Intuitivamente, $t(n) \in \Omega(f(n))$ significa que la función $t(n)$ crece **al menos tan rápido como** $f(n)$ cuando n tiende a infinito. Puede crecer más rápido, pero no más lento.

Orden Theta (Θ)

- La definición de Orden Theta combina las definiciones de Orden O y Orden Omega:
- $\Theta(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^+ \mid \exists c_1, c_2 \in \mathbb{R}^+,\ n_0 \in \mathbb{N} \mid c_1 \cdot f(n) \leq t(n) \leq c_2 \cdot f(n), n \geq n_0\}$
  - Es decir, es el conjunto de funciones que están acotadas **tanto superior como inferiormente** por $f(n)$.
  - En este sentido, $t(n)$ crece de manera "similar" a $f(n)$.
- Intuitivamente, $t(n) \in \Theta(f(n))$ significa que la función $t(n)$ crece **exactamente al mismo ritmo que** $f(n)$ cuando n tiende a infinito, salvo constantes multiplicativas. Solo se cumple si tanto $t(n) \in O(f(n))$ como $t(n) \in \Omega(f(n))$ se cumplen a la vez.

Propiedades

- $g(n) \in \Omega(f(n)) \leftrightarrow f(n) \in O(g(n))$ (Regla de dualidad)
- $g(n) \in \Theta(f(n)) \leftrightarrow [g(n) \in O(f(n)) \land g(n) \in \Omega(f(n))]$
- $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$
- $f(n) \in O(f(n))$ (Reflexividad)
- $(f(n) \in O(g(n)) \land g(n) \in O(h(n))) \to f(n) \in O(h(n))$ (Transitividad)

Regla del Umbral

- El umbral n_0 en las definiciones de Orden O, Omega y Theta puede resultar útil pero nunca es necesario cuando se consideran funciones estrictamente positivas, es decir $t, f : \mathbb{N} \to \mathbb{R}^+$.
- La Regla del Umbral dice: $f, t : \mathbb{N} \to \mathbb{R}^+, t(n) \in O(f(n)) \leftrightarrow \exists c \in \mathbb{R}^+ \mid t(n) \leq c \cdot f(n) \quad \forall n \in \mathbb{N}$

Regla del Máximo

- Sean f y g dos funciones tales que $f, g : \mathbb{N} \to \mathbb{R}^+$.
- $O(f(n) + g(n)) = O(\max(f(n), g(n)))$
- Es decir, el orden de la suma de dos funciones es igual al orden de la función que crece más rápido entre las dos.
- Vale para Θ también.
- Se puede generalizar a N funciones.

Regla del Límite

- La notación asintótica tiene relación con la idea de crecimiento arbitrario de la entrada y de cómo se comportan las funciones en el límite.
- Sea f y g dos funciones tales que $f, g : \mathbb{N} \to \mathbb{R}^+$.
- Las reglas son:
  - Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R}^+$ se cumple:
    - $f(n) \in O(g(n))$
    - $f(n) \in \Theta(g(n))$
    - $g(n) \in O(f(n))$
    - Intuitivamente, esto dice que las funciones crecen al mismo ritmo, ya que la razón entre ambas tiende a una constante positiva.
  - Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ se cumple:
    - $f(n) \in O(g(n))$
    - $f(n) \notin \Theta(g(n))$
    - $g(n) \notin O(f(n))$
    - Intuitivamente, esto dice que $f(n)$ crece igual o más lento que $g(n)$, ya que la razón entre ambas tiende a cero.
  - Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ se cumple:
    - $f(n) \notin O(g(n))$
    - $f(n) \in \Omega(g(n))$
    - $f(n) \notin \Theta(g(n))$
    - $g(n) \in O(f(n))$
    - Intuitivamente, esto dice que $f(n)$ crece igual o más rápido que $g(n)$, ya que la razón entre ambas tiende a infinito.

Complejidad Temporal

Máquinas de Turing No Determinísticas (MTN)

Definición

- Una MTN es una máquina de Turing que dado un estado y un símbolo siendo leido por el cabezal, tiene un conjunto finito de posibles transiciones a seguir para el siguiente movimiento.
- La MTN acepta su entrada si cualquier sucesión de transiciones de movimiento se detiene en q_A.
- Formalmente: $M = \langle Q, \Sigma, \Gamma, \Delta, q_0, q_A, q_R \rangle$

    - Q: Conjunto finito de estados.
    - Σ: Alfabeto de la entrada.
    - Γ: Alfabeto de la cinta ($\Sigma \subseteq \Gamma$).
    - Δ: Función de transición ($\Delta : Q \times \Gamma \to \mathcal{P}(Q \cup \{q_A, q_R\} \times \Gamma \times \{D, I, S\})$).
    - q_0: Estado inicial ($q_0 \in Q$).
    - q_A: Estado de aceptación ($q_A \notin Q$).
    - q_R: Estado de rechazo ($q_R \notin Q$).
- Una MTN acepta su entrada w si y solo si existe al menos una computación a partir de w que alcanza el estado q_A.
- Para diferenciar, denotamos a la máquina de Turing determinística como MTD y a la máquina de Turing no determinística como MTN. Una MTD es un caso especial de MTN donde la función de transición Δ devuelve un único movimiento posible para cada par (estado, símbolo leído), es decir, un conjunto unitario.
- Este tipo de MT es equivalente en cuanto a poder de cómputo a la MTD, es decir, cualquier lenguaje que pueda ser reconocido por una MTN también puede ser reconocido por una MTD y viceversa. Sin embargo, la eficiencia en términos de tiempo de cómputo puede variar significativamente entre ambos modelos.

Ejemplo

- Construir una MTN M tal que $L(M) = \{w \in \{0,1,2,3\}^* \mid \text{w empieza con 00 o 01}\}$.
- Se define la función de transición Δ de la siguiente manera:
    - $\Delta(q_0, 0) = \{(q_1, 0, D), (q_2, 0, D)\}$ **No determinismo**
    - $\Delta(q_0, x) = \{(q_R, x, S)\}$ Para todo $x \in (\Gamma - \{0\})$
    - $\Delta(q_1, 0) = \{(q_A, 0, S)\}$
    - $\Delta(q_1, x) = \{(q_R, x, S)\}$ Para todo $x \in (\Gamma - \{0\})$
    - $\Delta(q_2, 1) = \{(q_A, 1, S)\}$
    - $\Delta(q_1, x) = \{(q_R, x, S)\}$ Para todo $x \in (\Gamma - \{1\})$

Traza de computación

- Para una MTN, la traza de computación tiene forma de árbol, donde cada rama representa una computación, es decir, una sucesión de alternativas de movimiento.
- En el ejemplo anterior, si $w = 021$, la traza de computación sería:
    - $q_0021 \vdash 0q_121 \vdash 0q_R21$
    - $q_0021 \vdash 0q_221 \vdash 0q_R21$
    - M rechaza a 021.
- Si $w = 0054$, la traza de computación sería:
    - $q_00054 \vdash 0q_1054 \vdash 0q_A054$
    - $q_00054 \vdash 0q_2054 \vdash 0q_R054$
    - M acepta a 0054 en el primer "universo", mientras que rechaza en el segundo. **Como existe al menos un universo donde acepta, entonces M acepta a 0054, sin importar que en otro universo lo rechace o incluso loopee**.

Aceptación

- Una MTN acepta una cadena de entrada w si existe al menos una secuencia de transiciones que lleva al estado de aceptación q_A.
- Es decir, no importa que algunas secuencias lleven al estado de rechazo q_R o que algunas computaciones nunca terminen (loopeen) siempre y cuando haya al menos una secuencia que termine en q_A, la MTN acepta la cadena.

Equivalencia

- Si L es un lenguaje aceptado por una MTN M_1, entonces L es aceptado por una MTD M_2.
- Esto se demuestra usando el famoso recorrido BFS (Breadth-First Search) en el árbol de computación de la MTN M_1.

Complejidad Temporal

De una MTD

- Sea M una MTD con k cintas.
- M es de complejidad temporal $t(n)$ si para toda cadena de entrada w de longitud n, M hace a lo sumo $t(n)$ movimientos antes de detenerse (aceptar o rechazar).

De una MTN

- Sea M una MTN con k cintas.
- M es de complejidad temporal $t(n)$ si para toda cadena de entrada w de longitud n, **todas** las posibles computaciones de M hacen a lo sumo $t(n)$ movimientos antes de detenerse (aceptar o rechazar).

Clases de Complejidad

DTIME

- Sea L un lenguaje.
- Decimos que $L \in DTIME(t(n))$ si y solo si existe una MTD M con $L = L(M)$ tal que la complejidad temporal de M pertenece a $O(t(n))$.

NTIME

- Sea L un lenguaje.
- Decimos que $L \in NTIME(t(n))$ si y solo si existe una MTN M con $L = L(M)$ tal que la complejidad temporal de M pertenece a $O(t(n))$.
- $DTIME(t(n)) \subseteq NTIME(t(n))$ siempre, ya que una MTD es un caso especial de MTN.

P

- La clase de lenguajes P está formada por todos los lenguajes que pueden ser decididos por una MTD en **tiempo polinomial**.
- Formalmente: $P = \bigcup_{i \geq 0} DTIME(n^i)$
- Todo lenguaje en P es decidible, ya que las MTD siempre se detienen.

NP

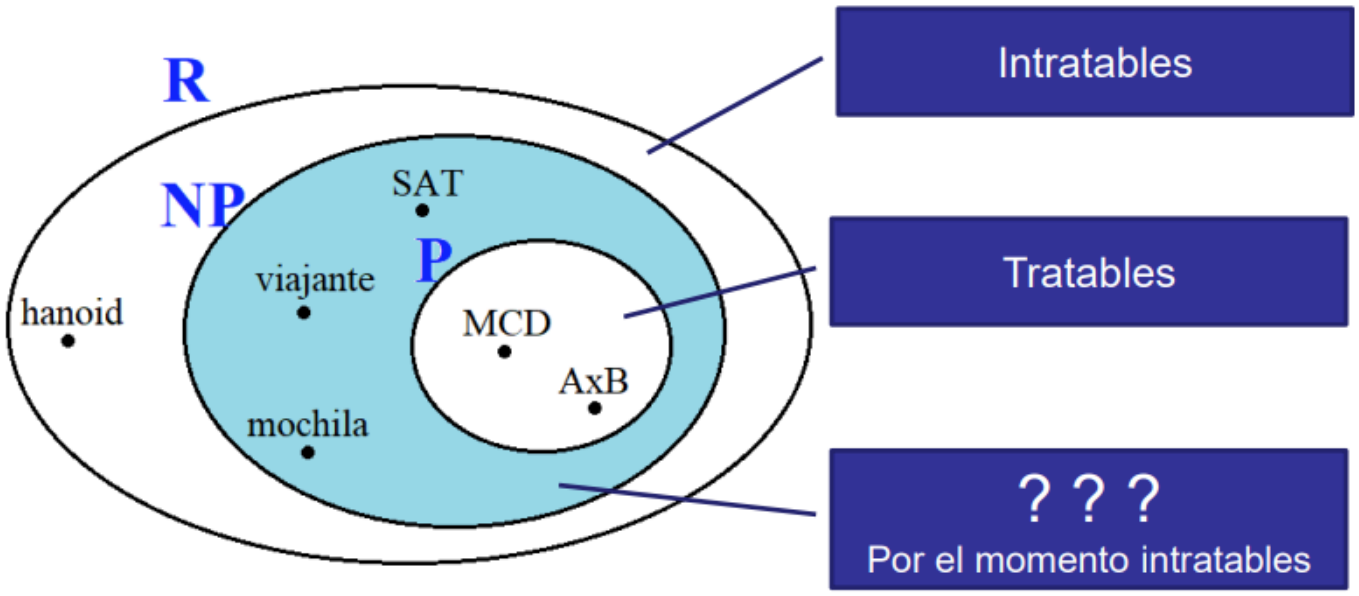
- La clase de lenguajes NP está formada por todos los lenguajes que pueden ser decididos por una MTN en **tiempo polinomial**.
- Formalmente: $NP = \bigcup_{i \geq 0} NTIME(n^i)$
- Claramente, $P \subseteq NP$, ya que todas las MTD son también MTN.
- Todo lenguaje en NP es decidible, ya que las MTN en tiempo polinomial siempre se detienen.

Teorema

- Si L es un lenguaje recursivo que es aceptado por una MTN M_1 en tiempo $t(n)$, entonces existe una MTD M_2 que acepta a L en tiempo menor o igual que $d^{t(n)}$ con d una constante mayor que 1 que depende de M_1.
- Es decir, para todo lenguaje que es aceptado por una MTN en $t(n)$, existe una MTD que lo acepta en tiempo exponencial $d^{t(n)}$.
- Se sabe que existen lenguajes recursivos que NO pertenecen a NP, es decir, $(R - NP) \neq \emptyset$ pero no se sabe si $NP = P$.

Tratabilidad

- En general se considera que la clase P representa los problemas "tratables". La mayoría de los problemas de esta clase tienen algoritmos con una implementación razonablemente eficiente. (Máximo Común Divisor, Multiplicación de matrices, 2-SAT, etc).
- Sin embargo existen ciertos problemas en P que no son tratables debido a que:
  - El grado del polinomio es muy alto (por ejemplo $O(n^{30})$).
  - Las constantes ocultas de la notación asintótica son muy grandes.
  - La demostración de pertenencia a P no es constructiva y no se conoce un algoritmo eficiente para resolver el problema en la práctica.
- El mapa de la tratabilidad es el siguiente:



- El punto principal es que la brecha es inaceptablemente grande pues existen muchos problemas importantes que están en NP a los que no se les conoce una solución tratable, pero tampoco se ha demostrado que esta solución no exista.

Problema del Viajante de Comercio (TSP)

- Tenemos un gráfico de ciudades.
- Hay un comerciante.
- El comerciante tiene que recorrer todas las ciudades partiendo desde una y volviendo a la misma pasando por todas las ciudades sin repetir ninguna e intentando minimizar la distancia total recorrida, es decir, que la distancia total recorrida no sea mayor que una constante d.
- Este problema se puede resolver con una MTN:
  - El input w contiene:
    - La lista de m ciudades (identificadas con números del 1 al m)
    - Las distancias de los caminos entre las ciudades.
    - La longitud máxima d permitida.
  - Se genera de forma no determinística las $m!$ trayectorias, que son todas las posibles permutaciones de las m ciudades.
    - Se escribe una permutación cualquiera de los números entre 1 y m en la cinta. Se escriben dos listas, la primera con los números ordenados de 1 a m y la segunda vacía. Se elige de manera no determinística uno de la primera lista que se tacha y pasa a la segunda. Cada ciclo tiene un costo de $O(m)$ y se repite m veces hasta completar la permutación $(O(m^2))$, como la lista de ciudades es parte de la entrada podemos acotarlo con $O(n^2)$.
  - Se calcula la distancia del recorrido elegido. Suponiendo un costo $(O(n)$ para localizar en la entrada de la distancia entre 2 ciudades cualquiera, el costo de calcular la distancia del recorrido es $O(n^2)$.
  - Si la distancia del recorrido es $\leq d$, para en q_A, sino para en q_R.
  - M trabaja en tiempo polinomial $O(n^2)$.

CO-NP

Definición

- Sea L un lenguaje.
- Decimos que $L \in \text{CO-NP}$ si y solo si el complemento de L pertenece a NP.
- Es decir, $L \in \text{CO-NP} \leftrightarrow \overline{L} \in NP$
- No se sabe cómo están relacionados NP y CO-NP. No se puede asumir que son iguales.

Teorema 1

- $L \in P \implies L \in (NP \cap \text{CO-NP})$
- Si un lenguaje está en NP, no se puede saber si su complemento también está en NP.
- Demostración:
  - $L \in P \implies L \in NP$ (Por $P \subseteq NP$)
  - $L \in P \implies \overline{L} \in P$ (Intercambiando estados de aceptación y rechazo en la MTD que decide a L)
  - $\overline{L} \in P \implies \overline{L} \in NP$ (Por $P \subseteq NP$)
  - $\overline{L} \in NP \implies L \in \text{CO-NP}$ (Por def. de CO-NP)

Teorema 2

- Hipotéticamente, si $NP \neq \text{CO-NP}$, entonces $P \neq NP$.
- Demostración:
  - Si $P = NP \implies NP = \text{CO-NP}$
  - Por contrarrecíproco: Si $NP \neq \text{CO-NP} \implies P \neq NP$

Reducción Polinomial

Definición

- Sean L_1 y L_2 dos lenguajes sobre un alfabeto Σ.
- Decimos que L_1 se reduce polinomialmente a L_2, denotado como $L_1 \ \alpha_p \ L_2$, si y solo si $L_1 \ \alpha \ L_2$ y además la función de reducción f es computada por una MTD que trabaja en tiempo polinomial, es decir, $(f \in P)$.

Teorema 3

- Sean L_1 y L_2 dos lenguajes.
- Si $L_1 \ \alpha_p \ L_2$ y además $L_2 \in P$ entonces $L_1 \in P$.

Teorema 4

- Sea L un lenguaje tal que $\emptyset \subset L \subset \Sigma^*$.

- Entonces para cualquier lenguaje L' con $L' \in P$, se tiene que $L' \alpha_p L$.

NP-Hard

- Sea L un lenguaje.
- $L \in NPH$ si para todo lenguaje $L' \in NP$, se cumple que $L' \alpha_p L$.

NP-Completo

Definición

- Sea L un lenguaje.
- $L \in NPC$ si y solo si:
  - $L \in NPH$
  - $L \in NP$

Teorema 5

- Si existe un lenguaje $L \in NPC$ tal que $L \in P$, entonces $P = NP$.
- Hasta ahora no se ha encontrado ningún lenguaje que cumpla lo anterior.
- Según este teorema, para demostrar que $P = NP$ alcanzaría con encontrar una solución polinomial para cualquiera de los problemas NPC conocidos.

Teorema 6

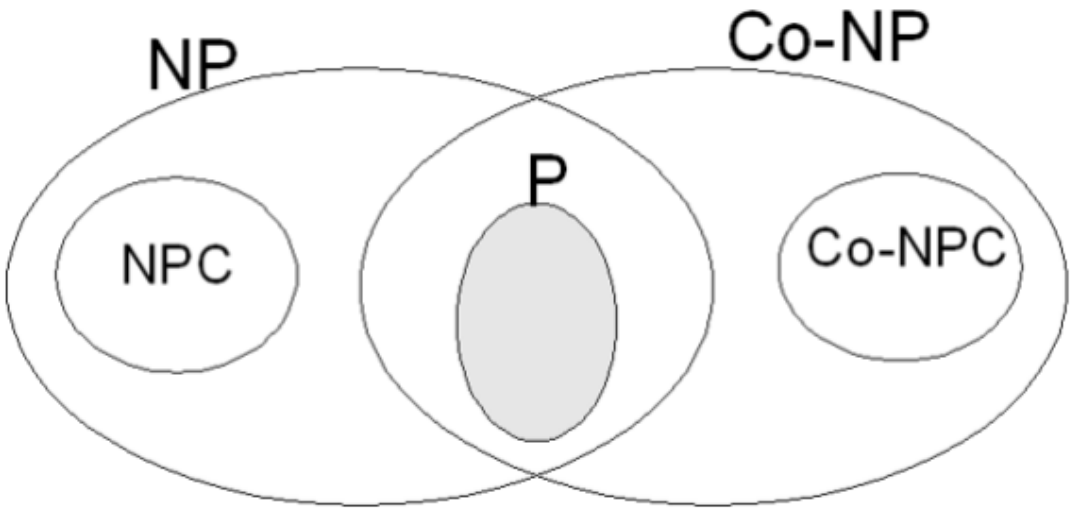
- Sean L_1 y L_2 dos lenguajes tal que $L_1 \in NP$ y $L_2 \in NP$.
- Si $L_1 \in NPC$ y $L_1 \alpha_p L_2$, entonces $L_2 \in NPC$.

Ejemplo de lenguaje NPC : SAT

- **Literal**: Variable proposicional o su negación.
- **FNC**: Forma normal conjuntiva, es una fórmula lógica que es una conjunción de cláusulas, donde cada cláusula es una disyunción de literales. Por ejemplo: $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$
- **Enunciado del problema SAT**: Dada una fórmula proposicional en FNC, determinar si es satisfactible, es decir, si existe alguna asignación de valores de verdad a las variables que haga que la fórmula tenga el valor de verdad verdadero.
- $SAT = \{\varphi \mid \varphi \text{ es una fórmula booleana en FNC satisfactible}\}$
- Se ha demostrado que $SAT \in NPC$ vía el teorema de Cook/Levin.

$P = NP$?

- Por varias décadas de estudio los investigadores no han podido determinar si la afirmación $P = NP$ es verdadera o falsa.
- Se cree ampliamente que $P \neq NP$, es decir, que existen problemas en NP que no pueden ser resueltos en tiempo polinomial por una MTD, pero hasta ahora no se ha encontrado una prueba formal que lo demuestre.
- La resolución de esta cuestión es uno de los problemas más importantes y desafiantes en la teoría de la computación y tiene implicaciones significativas en campos como la criptografía, la optimización y la inteligencia artificial.
- Mapa completo de P, NP, NPC, CO-NPC y CO-NP:



Clasificación completa de todos los lenguajes vistos en la materia

Lenguaje	$\in RE$	$\in R$	$\in$ CO-RE	$\in P$	$\in NP$	$\in$ CO-NP	$\in NPH$	$\in NPC$
$\emptyset$	✅	✅	✅	✅	✅	✅	❌	❌
$\{\lambda\}$	✅	✅	✅	✅	✅	✅	❌	❌
Σ^*	✅	✅	✅	✅	✅	✅	❌	❌
L_D	❌	❌	✅	❌	❌	?	✅	❌
$\overline{L_D}$	✅	❌	❌	❌	❌	?	?	?
L_u	✅	❌	❌	❌	❌	?	?	?
$\overline{L_u}$	❌	❌	❌	❌	❌	?	?	?
L	❌	❌	❌	❌	❌	?	?	?

Lenguaje	$\in RE$	$\in R$	$\in$ CO-RE	$\in P$	$\in NP$	$\in$ CO-NP	$\in NPH$	$\in NPC$
HP	✅	❌	❌	❌	❌	❓	✅	❌
$\overline{HP}$	❌	❌	❌	❌	❌	❓	❓	❓
L_{Σ^*}	❌	❌	❌	❌	❌	❓	❓	❓
L_{EQ}	❌	❌	❌	❌	❌	❓	❓	❓
TSP	✅	✅	✅	❌	✅	❓	✅	✅
SAT	✅	✅	✅	❌	✅	❓	✅	✅

Análisis de Algoritmos

Asociación de conceptos vistos a lo largo de la materia

- Un problema computacional se puede asociar a una Máquina de Turing Determinística.
- Una entrada a un problema se puede asociar a una instancia específica del problema computacional.
- Los problemas computables se pueden asociar a los lenguajes decidibles (R).
- Una Máquina de Turing Determinística se puede asociar a un algoritmo.
- Computable no es lo mismo que Tratable.
- La complejidad temporal es principalmente un análisis y clasificación de los distintos tipos de lenguajes decidibles.
- Un problema computable tiene infinitas formas de ser resuelto, es decir, infinitas Máquinas de Turing Determinísticas que lo resuelven. Obviamente, algunas formas de resolverlo son mejores que otras.

Análisis de algoritmos

Definición

Consiste en el estudio teórico de la eficiencia de los algoritmos, generalmente en términos del uso de los recursos de hardware. La eficiencia se mide en qué cantidad de recursos (tiempo y espacio) utiliza un algoritmo para resolver un problema en función del tamaño de la entrada.

Utilidad

- Permite comparar diferentes algoritmos de forma puramente teórica, sin necesidad de implementarlos y ejecutarlos en hardware real.
- Permite analizar la escalabilidad de los algoritmos, es decir, cómo se comportan a medida que el tamaño de la entrada crece.
- La matemática algorítmica provee un lenguaje de comportamiento abstracto que permite razonar sobre los algoritmos sin necesidad de entrar en detalles de implementación específicos.

Principio de Invarianza

- Dos implementaciones diferentes del mismo algoritmo difieren en eficiencia a lo sumo en una constante multiplicativa.
- Intuitivamente, esto significa que si tenemos dos versiones de un mismo algoritmo, una puede ser más rápida que la otra, pero la diferencia en tiempo de ejecución no crecerá desproporcionadamente a medida que aumente el tamaño de la entrada. La eficiencia relativa entre las dos implementaciones se mantendrá constante.

Tiempo vs espacio

- Sea A un algoritmo que resuelve un problema computacional.
- El **tiempo de ejecución $t_A(n)$ de A** es la cantidad de pasos, operaciones, o acciones elementales que debe realizar el algoritmo al ser ejecutado con una entrada de tamaño n.
- El **espacio de ejecución $e_A(n)$ de A** es la cantidad de datos elementales que el algoritmo necesita al ser ejecutado en una instancia de tamaño n, sin contar la representación de la entrada.
- Ambas definiciones son claramente ambiguas, porque no se especifica claramente cuáles son las operaciones o datos elementales. Además, dado que existe más de una instancia de tamaño n, no está claro cuál de todas ellas es la que se tiene en cuenta para el análisis.

Operaciones elementales

- El tiempo de ejecución está cotado por una constante que depende solo de la implementación usada.
- Las operaciones elementales no dependen del tamaño de la entrada: solo interesa el número de operaciones elementales realizadas.
- En general, las sumas, multiplicaciones, asignaciones, etc, son consideradas operaciones elementales.

Tipos de análisis

- **Peor caso**: Se analiza la instancia de tamaño n que hace que el algoritmo tarde más tiempo en ejecutarse.
- **Caso promedio**: Se analiza el tiempo de ejecución promedio del algoritmo sobre todas las instancias de tamaño n .
- **Mejor caso**: Se analiza la instancia de tamaño n que hace que el algoritmo tarde menos tiempo en ejecutarse.

- **Probabilístico**: Se analiza el tiempo de ejecución esperado del algoritmo considerando que la entrada es generada aleatoriamente según una distribución de probabilidad dada.

Uso de la notación asintótica

En el contexto de análisis de algoritmos, la notación asintótica es útil por varias razones:

1. Caracteriza en forma simple a la eficiencia o uso de recursos de los algoritmos.
2. Se independiza de detalles de implementación específicos.
3. Analiza el comportamiento de las funciones en el límite cuando la entrada crece arbitrariamente.

Temas principales a considerar

1. Estructuras de control

Secuencias

- Sean $P1$ y $P2$ dos procesos secuenciales.
- $P1$ tiene un tiempo de ejecución $t_1(n)$.
- $P2$ tiene un tiempo de ejecución $t_2(n)$.
- El tiempo de ejecución total del proceso secuencial $P1; P2$ es: $t_{1;2}(n) = t_1(n) + t_2(n)$
- En términos de notación asintótica se usa la Regla del Máximo:
  - $t_{1;2}(n) = t_1(n) + t_2(n) \in O(\max(t_1(n), t_2(n)))$
  - $t_{1;2}(n) = t_1(n) + t_2(n) \in \Theta(\max(t_1(n), t_2(n)))$
- Intuitivamente, el tiempo de ejecución total de dos procesos secuenciales es igual a la suma de sus tiempos de ejecución individuales.
- Esto se puede generalizar a N procesos secuenciales.
- Esto se puede aplicar no solo a procesos enteros sino también a instrucciones individuales.

Condicional

- La estructura típica del condicional es:

```
if (cond)
{
    ...
}
else
{
    ...
}
```

- Se puede definir:

- $t_1(n)$: Tiempo de ejecución para evaluar la condición.
- $t_2(n)$: Tiempo de ejecución del bloque "then".
- $t_3(n)$: Tiempo de ejecución del bloque "else".

- Para analizarlo, siempre se considera el peor caso:
  - $t_{if}(n) = t_1(n) + \max(t_2(n), t_3(n))$
  - O más directamente: $\max(t_1(n), t_2(n), t_3(n))$

Iteraciones uniformes

- La estructura típica de una iteración uniforme es:

```
for (i = 0; i < N; i++)
{
    ...
}
```

- En el inicio del `for`, siempre se sabe de antemano cuántas veces se va a ejecutar el cuerpo del ciclo.
- Para analizar el tiempo total, se debe saber el tiempo de ejecución del cuerpo del ciclo, al que llamamos $t(i)$.
- Normalmente hay dos alternativas: que el tiempo del cuerpo dependa de i o que no dependa de i.
  - Si no depende de i, es decir, $t(i) = t$, entonces el tiempo total es: $t_{for}(n) = (N \cdot t)$
  - Si depende de i, entonces el tiempo total es: $t_{for}(n) = \sum_{i=0}^{N-1} t(i)$
    - No hay un mejor o peor caso, ya que el número de iteraciones es fijo.
- Más sumatorias útiles:
  - $\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$
  - $\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6}$
- Todo lo anterior vale si y solo si $1 \leq N$, es decir, que el valor de inicio sea menor o igual que el valor final.

Iteraciones no uniformes

- La estructura típica de una iteración no uniforme es:

```
while (cond)
{
    ...
}

do
{
    ...
} while (cond)
```

- En este caso, no se sabe de antemano cuántas veces se va a ejecutar el cuerpo del ciclo.
- Existen dos formas de analizar estas estructuras:
  - Funciones de variables que decrece.

- Recurrencias.

2. Barómetro

- Una instrucción barómetro es una instrucción que se ejecuta al menos tantas veces como cualquier otra instrucción excepto quizás una cantidad constante de veces.
- Si $t(n)$ es el tiempo del algoritmo a analizar, $t(n)$ es $\Theta(f(n))$ donde $f(n)$ es la cantidad de veces que se ejecuta la instrucción barómetro.

3. Análisis del caso promedio

- Requiere hacer un uso implícito/explícito de distribución de probabilidades de las instancias de tamaño n.

4. Análisis amortizado

- Es útil cuando:
  - Es muy poco probable que en todas las llamadas a una función se tenga siempre el peor caso.
  - La cantidad de operaciones está relacionada con una secuencia de uso de un algoritmo.
- Ejemplos:
  - Estructuras de datos, inserción en un grafo.
- Es muy específico a ciertos casos/algoritmos.
- Es una alternativa al análisis del caso peor/mejor/promedio.
- En términos simples, la cantidad de veces que se realiza una operación depende del estado actual de la estructura de datos.
- Normalmente se busca un promedio del $t(n)$ en llamadas sucesivas y no independientes.
- Existen tres método principales:
  - Agregado.
  - Contable.
  - Potencial.

5. Recurrencias

Concepto

- Son funciones de tiempo que están definidas en función de sí mismas.
- Describen una secuencia de números, donde el o los término/s inicial/es (cantidad finita) son dados de forma explicita y el resto de los términos se definen en función de los términos anteriores.
- Se los suele asociar a la estrategia Divide and Conquer y a la estrategia Divide-Solve/Conquer-Combine.
- En cualquier caso, se asocian a algoritmos recursivos.

Ejemplo

- El ejemplo clásico es el cálculo del factorial de un número n:

```
int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

- El tiempo de ejecución $t(n)$ de este algoritmo se puede definir como:
  - $t_{fac}(n) = \begin{cases} 1 & \text{si } n = 0 \\ t_{fac}(n-1) + 2 & \text{si } n > 0 \end{cases}$
  - El $+2$ se debe a que se tiene dos operaciones, primero una resta `n - 1` y luego una multiplicación `n * factorial(n - 1)` .
- Análisis:
  - $t_{fac}(n) = t_{fac}(n-1) + 2$
  - $t_{fac}(n) = t_{fac}(n-2) + 2 + 2$
  - $t_{fac}(n) = t_{fac}(n-3) + 2 + 2 + 2$
  - ...
  - $t_{fac}(n) = t_{fac}(0) + 2 + 2 + ... + 2$ (n veces 2)
  - $t_{fac}(n) = t_{fac}(0) + 2n$
  - $t_{fac}(n) = 1 + 2n$
  - Por lo tanto $t_{fac}(n) \in O(n)$.

Recurrencias homogéneas vs no homogéneas

- Una recurrencia es homogénea si la ecuación es igual a cero si se pasan todos los términos $t(n)$ a un mismo lado de la igualdad.
  - Por ejemplo: $t(n) = 2t(n-1) + 3t(n-2)$ es homogénea porque se puede escribir como $t(n) - 2t(n-1) - 3t(n-2) = 0$.
- Una recurrencia es no homogénea si la ecuación es igual a algo distinto de cero si se pasan todos los términos $t(n)$ a un mismo lado de la igualdad.
  - Por ejemplo: $t(n) = 2t(n-1) + 1$ es no homogénea porque se puede escribir como $t(n) - 2t(n-1) = 1$.

Recetas para resolver recurrencias

1. $t(n) = a \cdot t(n-b) + f(n)$ (reduce restando)
   i. $t(n) = \begin{cases} c & \text{si } n \leq b \\ a \cdot t(n-b) + f(n) & \text{si } n > b, \text{con } f(n) \in O(n^k) \end{cases}$

   ii. $t(n) \in \begin{cases} O(n^{k+1}) & \text{si a} = 1 \\ O(a^{\text{n div b}}) & \text{si a} > 1 \end{cases}$
   iii. Vale con todos los $\Theta()$.
2. $t(n) = a \cdot t(\frac{n}{b}) + f(n)$ (reduce dividiendo)
   i. $t(n) = \begin{cases} c & \text{si } n \leq b \\ a \cdot t(\frac{n}{b}) + f(n) & \text{si } n > b, \text{con } f(n) \in O(n^k) \end{cases}$
   ii. $t(n) \in \begin{cases} O(n^{log_b a}) & \text{si } a > b^k \\ O(n^k \cdot log_b n) & \text{si } a = b^k \\ O(n^k) & \text{si } a < b^k \end{cases}$
   iii. Vale con todos los $\Theta()$.

- a es la cantidad de llamadas recursivas.
- b es una constante.
- $f(n)$ son operaciones extra a las llamadas recursivas.
- Estas dos recetas no cubren todos los casos posibles de recurrencias, pero sí los más comunes.

6. Estructuras de datos

- La mayoría son recursivas por naturaleza.
- En algunos casos sirve el análisis amortizado para estructuras de datos.

7. Diseño + Análisis

Concepto

- Mucho más asociado al diseño que al análisis.
- Las alternativas más comunes son:
  - Divide and Conquer.
  - Greedy.
  - Programación Dinámica.
  - Algoritmos probabilísticos.

Divide and Conquer

- Se relaciona directamente con la recursión y por lo tanto con las recurrencias ya vistas.
- Básicamente, consiste en dividir el problema en subproblemas más pequeños, resolver esos subproblemas de forma recursiva y luego combinar las soluciones de los subproblemas para obtener la solución del problema original.
- Ejemplos clásicos:
  - Merge Sort.
  - Quick Sort.

- Búsqueda binaria.

Greedy

- Se aplica principalmente a problemas de optimización, construyendo la solución paso a paso de manera iterativa.
- En cada paso, se elige una o varias decisiones, dependiendo del estado de avance/solución actual.
- Las elecciones se hacen basándose en un criterio local que parece ser el mejor en ese momento.
- Se llama greedy justamente porque el algoritmo "toma lo que parece mejor" en cada paso sin considerar las consecuencias futuras.
- El ejemplo clásico es el Traveling Salesman Problem:
  - Se elige la primer ciudad al azar.
  - Luego se elige la ciudad más cercana a la que se visitó en el paso anterior.
  - Se repite hasta visitar todas las ciudades.
  - Es decir, siempre se elige viajar a la ciudad más cercana relativa a la ciudad actual, sin considerar si esa elección es peor en el largo plazo.
- Los algoritmos de tipo Greedy siempre proveen una solución, pero no siempre es la solución óptima. Son muy sencillos de implementar.

Programación Dinámica

- Estrategia bottom-up, es decir, intenta resolver primero los subproblemas más pequeños y sencillos y luego utiliza esas soluciones para construir soluciones a subproblemas más grandes.
- El ejemplo clásico es el cálculo de la secuencia de Fibonacci:
  - Se calcula $F(0)$ y $F(1)$ directamente.
  - Luego se calcula $F(2) = F(0) + F(1)$.
  - Luego se calcula $F(3) = F(1) + F(2)$.
  - Y así sucesivamente hasta llegar a $F(n)$.
  - En este caso los "problemas sencillos" son los primeros dos términos de la secuencia de Fibonacci.

Algoritmos probabilísticos

- Asociados a problemas donde se deben tomar decisiones.
- En este tipo de algoritmos, estas decisiones se toman al azar.
- Hay tres tipos:
  - **Numéricos**: Usan intervalos de confianza sobre la respuesta.
  - **Monte Carlo**: Respuesta exacta con alta probabilidad, pero puede ser incorrecta.
  - **Las Vegas**: Respuesta exacta o sin respuesta.