# Computabilidad

## Máquina de Turing

- La definión de la máquina ideal.
- Más poderosa que las que tenemos hoy en día que están limitadas por la memoria.
- Lo tomaremos como la definición más general de algoritmo.

> ¿Qué relación existe entre las máquinas de Turing y la computabilidad de los problemas?
> ¿Son "computables" todos los problemas?
> ¿Son "igual de computables" todos los problemas computables?

## Computabilidad / Decidibilidad

- Si un problema tiene un algoritmo que siempre dice correctamente si una instancia de problema tiene respuesta "sí" o "no", entonces el problema se dice **"decidible"** o "computable" o "calculable".
- En otro caso, el problema se dice **"indecidible"** o "no computable" o "no calculable".
- En particular, si un problema indecidible tiene un algoritmo que siempre se detiene si una instancia de problema tienen respuesta y nunca se detiene si no tiene respuesta, entonces el problema se dice **"parcialmente computable"** o "parcialmente calculable".

## Ejemplo (Halting Problem)

Mostremos que el programa que:

- dado un programa $P$ y una entrada $E$;
- responde "sí" o "no", en función de si el programa $P$ se detiene con esa entrada $E$;

... no es decidible.

```
def halts(code, input):
    # do some magic tricks ...
    pass

def evil(code, input):
    if halts(code, input):
        while True:
            pass

def fatality():
    evil(evil, evil)
```

En cualquier caso, el programa `fatality` genera una contradicción, por lo que `halts` no pueden devolver un valor consistente en lo que respecta a la veracidad de que `evil` se detenga. Por tanto, `halts` no puede ser decidible.

# Ejemplo (Reducción)

Mostremos que el programa que:

- dado un programa y una entrada;
- responde "sí" o "no", en función de si el programa imprime "hola mundo" con esa entrada;

... no es decidible.

```
def prints_hello(code: str, input: str):
    # do some magic tricks ...
    pass

import os
import sys
def reduction(code: str):
    stdout = sys.stdout
    with open(os.devnull, "w") as sys.stdout:
        eval(code)
    sys.stdout = stdout
    print("Hello World!")

def halts(code: str):
    return prints_hello(reduction, code)
```

Si `prints_hello` fuera decidible, entonces acabamos de encontrar una solución que hace a `halts` decidible. Pero sabemos que `halts` no es decidible. Contradicción.

# Formalizando Definiciones

> ¿Cómo se traducen todas estas definiciones a Máquinas de Turing?

- **Lenguaje Recursivamente Enumerable:** un lenguaje se dice recursivamente enumerable si y solo si existe una máquina de Turing, de algún tipo, que lo reconoce.

  > Esta afirmación se aplica tanto a las Máquinas de Turing que aceptan cadenas por criterio de estado final, como a las que lo hacen por criterio de parada.

  - **Lenguaje Recursivo:** un lenguaje se dice recursivo, si existe una máquina de Turing con aceptación por criterio de estado final que lo reconoce.

    > Esto es lo más cercano a la definición informal de algoritmo (una secuencia de pasos bien definida que siempre termina y produce una respuesta).
    > Por tanto, los lenguajes recursivos denotan completa y exclusivamente los problemas decidibles / computables.

    - Algunas propiedades:
      - $L$ recursivo
        $\implies$ $L^c$ recursivo.

        - $L$ recursivamente enumerable & $L^c$ no recursivamente enumerable
          $\implies$ $L$ no recursivo

          > (porque de serlo, $L^c$ sería recursivo y por tanto recursivamente enumerable).

        - $L$ recursivamente enumerable & $L^c$ recursivamente enumerable
          $\implies$ $L$ recursivo
          $\implies$ $L^c$ recursivo.
    - La **diferencia** entre la familia de lenguajes **recursivamente enumerables** y los lenguajes **recursivos**, contiene los lenguajes para los que solo existen máquinas de Turing con aceptación por criterio de parada pero no por critero de estado final.

      > Estos lenguajes denotan los problemas parcialmente computables.

- **Lenguaje No Recursivamente Enumerable:** un lenguaje se dice no recursivamente enumerable si no existe máquina de Turing, de ningún tipo, que lo reconozca.

# Por qué Recursivamente Enumerable?

- Poder hablar de la $i$-ésima máquina de Turing ($M_i$),
- **¿Cómo codificar el alfabeto de cinta?**
  $0 \to 0, \quad 1 \to 00, \quad B \to 000, \quad \ldots$
- **¿Cómo codificar las transiciones?**
  $\delta(q_i, X_j) = (q_k, X_l, D_m) \implies 0^i 1 0^j 1 0^k 1 0^l 1 0^m$

  > con $i$, $j$, $k$, $l$ y $m$ siempre mayores que $0$.

- **¿Cómo codificar la máquina de Turing?**
  $C_1 11 C_2 11 C_3 11 \ldots C_{n-1} 11 C_n$
- Ordenar por longitud, y para cadenas con la misma longitud, ordenar lexicográficamente.

# Diagonalization Language ($L_d$)

- "*¿Acepta esta máquina de Turing su propio código como entrada?*"
- $L_d$ es un lenguaje sobre el alfabeto de entrada $\{0, 1\}$.
- $L_d$ es el conjunto de todas las cadenas $w_i$ tales que $w_i \notin L(M_i)$.
  - $L_d$ contiene todas las cadenas $w$ tal que la máquina de Turing $M$ cuyo código es $w$ no acepta su propio código $w$ como entrada.

- Lenguaje no recursivamente enumerable.
- La intuición va de que $L_d$ es el complemento de la diagonal de la matriz $M \times M$ de características. Al ser el complemento de la diagonal, no puede aparecer en la matriz, porque no es igual a ninguno de los que ya está. Para cualquier cantidad de $M$ filas y columnas, la diagonal puede tomar un valor entre $2^{|M|}$, que es mayor que $|M|$.
- Se demuestra asumiendo que es recursivamente enumerable, por tanto pertenece a la lista de todas las máquinas de Turing. Se llega a una contradicción viendo si $w_i$ pertence a $L_d$.

# Universal Language ($L_u$)

- "*¿Acepta esta máquina de Turing esta entrada en particular?*"

- $L_u$ es el conjunto de cadenas binarias que codifican una pareja $(M, w)$, donde $M$ is una máquina de Turing con alfabeto de entrada binario, y $w$ es una cadena binaria tal que $w \in L(M)$.
  - $L_u$ es el conjunto de cadenas que representan una máquina de Turing y una entrada aceptada por esa máquina de Turing.

- Lenguaje indecidible / no recursivo, pero recursivamente enumerable.

  > en terminos prácticos, no existe algoritmo que lo resuelva.

- Se demuestra que **es recursivamente enumerable** construyendo la máquina de Turing que simula $(M, w)$.
  - Por conveniencia, implementarlo multicinta.
  - En la primera cinta, el código de la máquina de Turing $M$.
  - En la segunda cinta, la cinta simulada de la máquina de Turing $M$ (cada símbolo codificado con $0$s y separados por un $1$).
    - El símbolo nulo de $M$ no se escribe inicialmente en la cinta, pero la máquina $U$ debe saber usar su propio símbolo nulo en las transiciones con símbolo $000$.
  - En la tercera cinta, el estado actual de la máquina de Turing $M$.
  - Simular las transiciones de $M$.

- Se demuestra que **no es recursivo** por reducción al absurdo (encontrando la contradicción porque el complemento de $L_u$ también tendría que ser recursivo y $L_d$ se podría reducir al complemento, pero ya sabemos que $L_d$ no es recursivamente enumerable y por tanto tampoco recursivo).