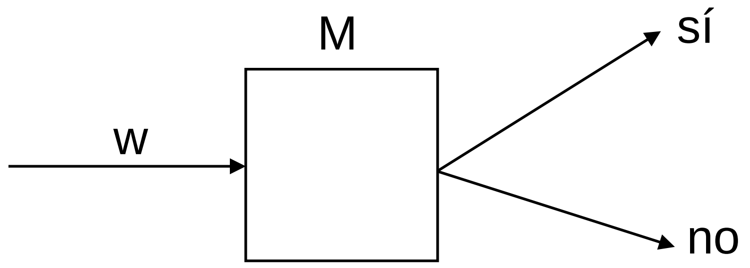
Clase teórica 3

Jerarquía de la computabilidad (segunda parte)

Tres posibilidades

1) Lenguajes con MT que los **aceptan y paran siempre.**

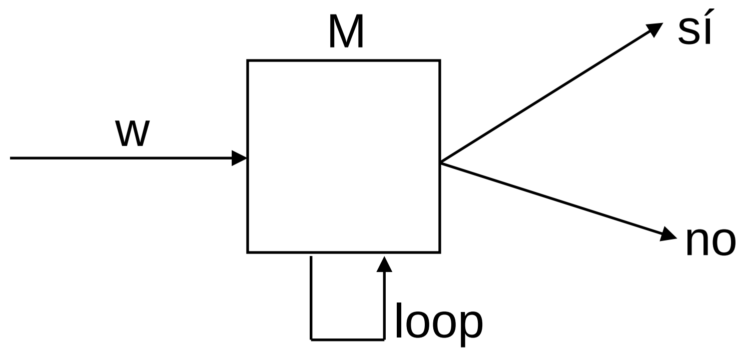


- Si w ∈ L(M), M acepta.
- Si w ∉ L(M), M rechaza.

Lenguajes **recursivos.**

Conjunto **R**.

2) Lenguajes con MT que los **aceptan pero no paran siempre.**



- Si w ∈ L(M), M acepta.
- Si w ∉ L(M), M rechaza o loopea.

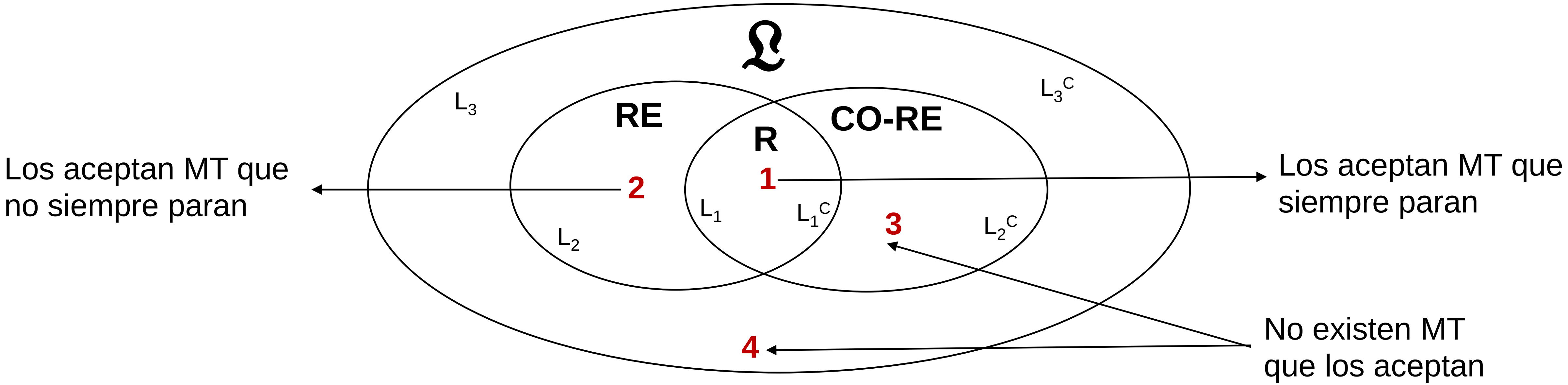
Lenguajes **recursivamente enumerables**.

Conjunto **RE**.

3) Lenguajes **sin MT que los acepten.**

Versión definitiva de la jerarquía de la computabilidad

- **Cuatro regiones de menor a mayor grado de dificultad:**



Probamos:

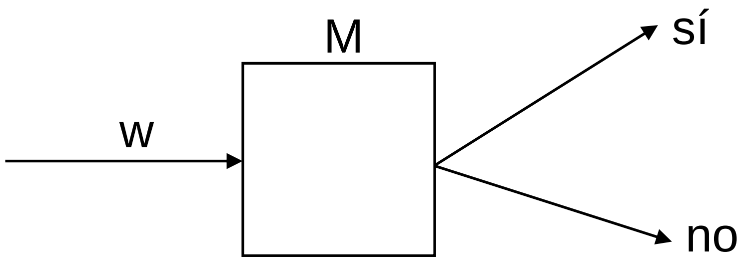
R = RE ∩ CO-RE

Falta probar:

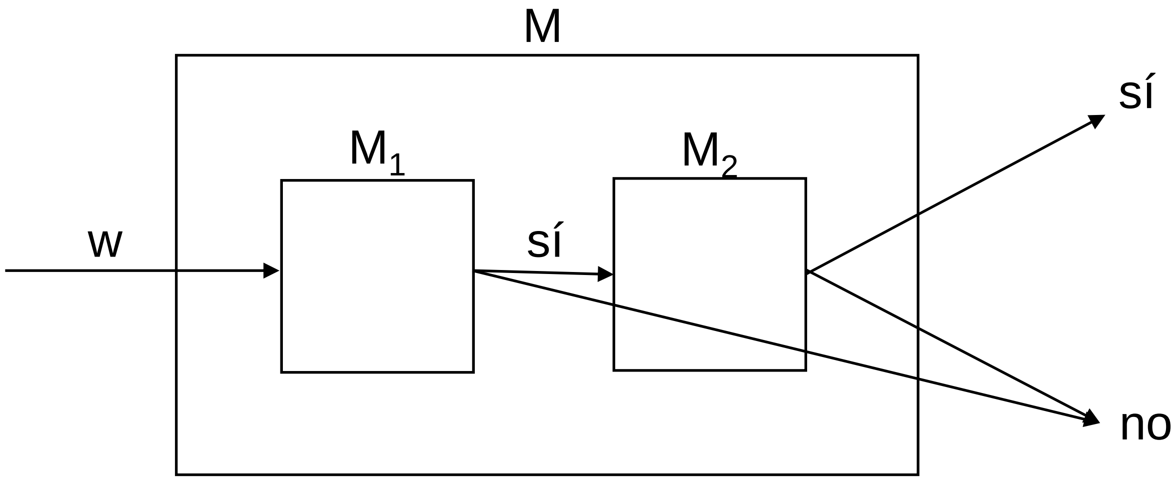
R ⊂ RE
RE ⊂ $\mathfrak{L}$
RE ∪ CO-RE ⊂ $\mathfrak{L}$

Pruebas constructivas para probar pertenencia a R o RE

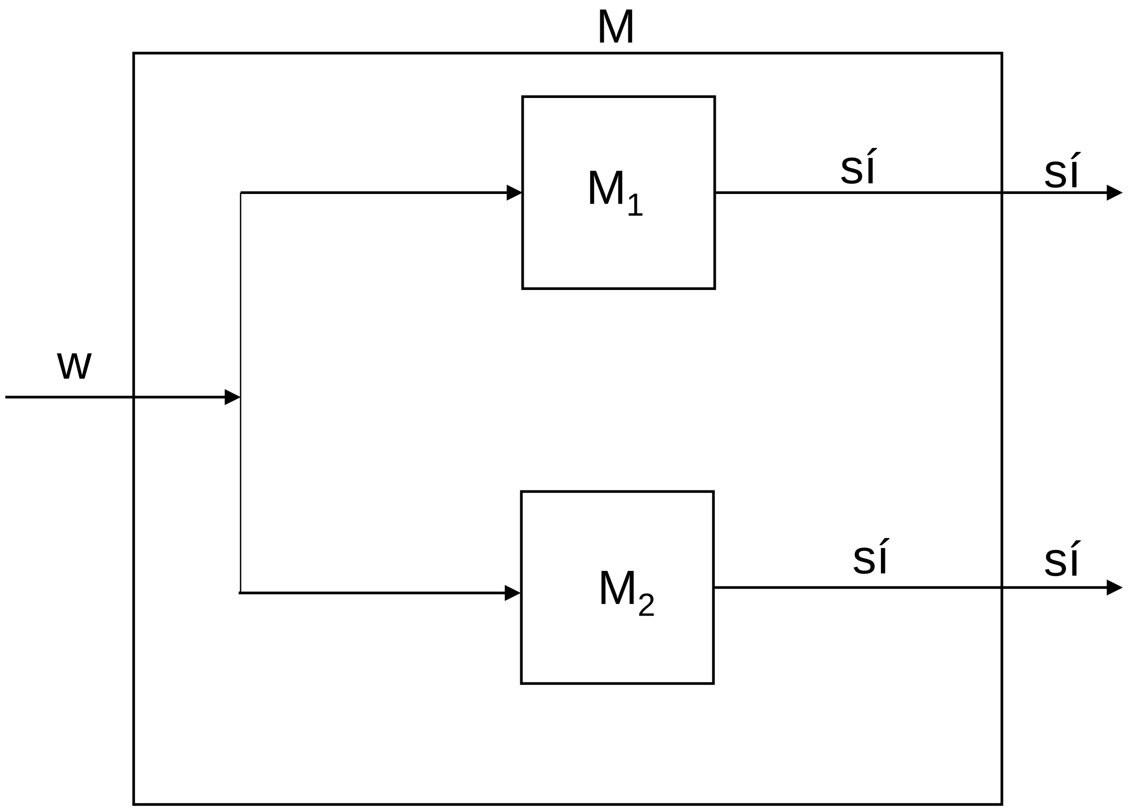
- **Ejemplos:**



M decide el lenguaje de las cadenas $a^n b^n$, con $n \geq 1$



M decide la intersección de los lenguajes que deciden M_1 y M_2



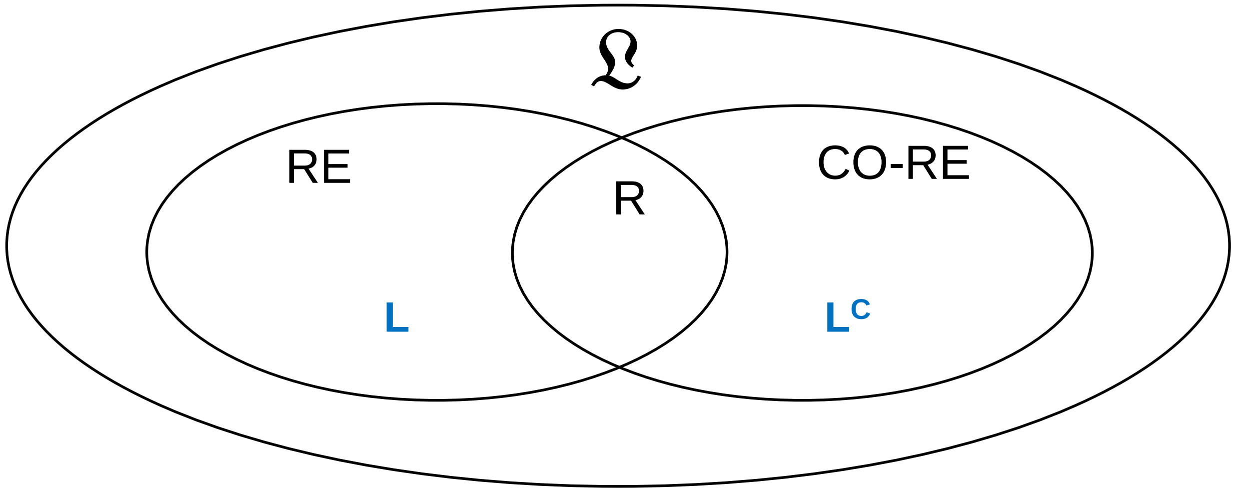
M reconoce la unión de los lenguajes que reconocen M_1 y M_2 (puede no parar)

Pruebas no constructivas para probar no pertenencia a R o RE

- Vamos a encontrar:

  Un primer lenguaje **L** en RE – R, para probar **R** $\subset$ **RE**

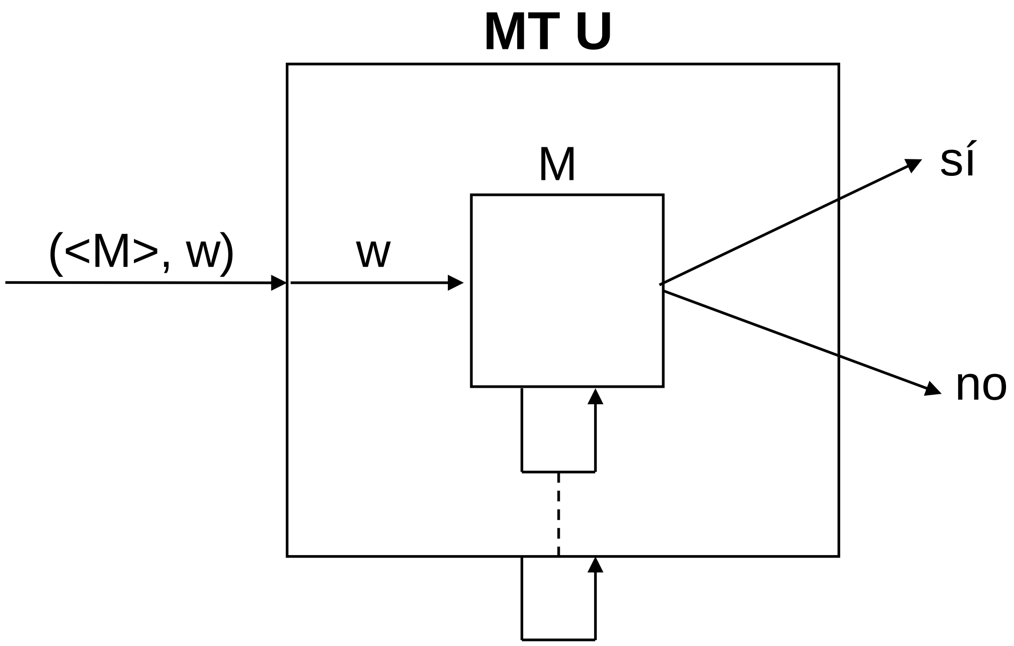
  Un primer lenguaje **L^C** en CO-RE – R, para probar **RE** $\subset \mathfrak{L}$

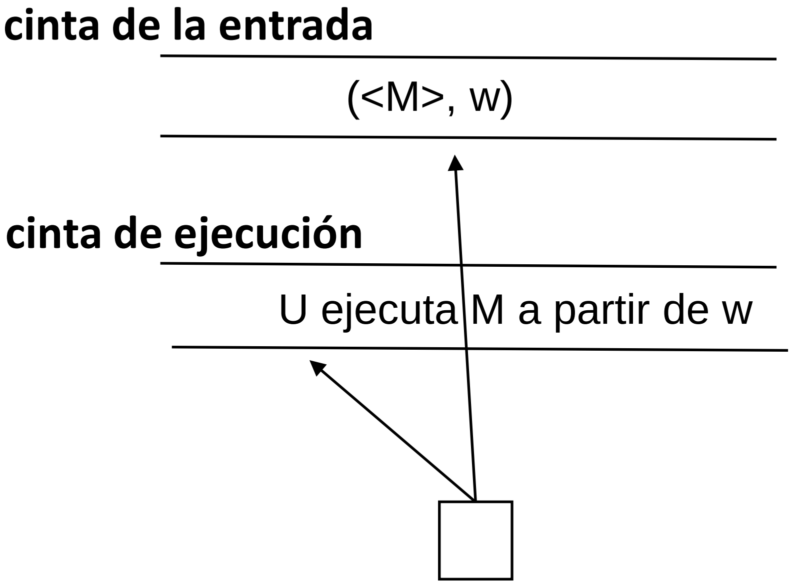


- Para ello, antes tenemos que introducir la **máquina de Turing universal.**

Máquina de Turing universal

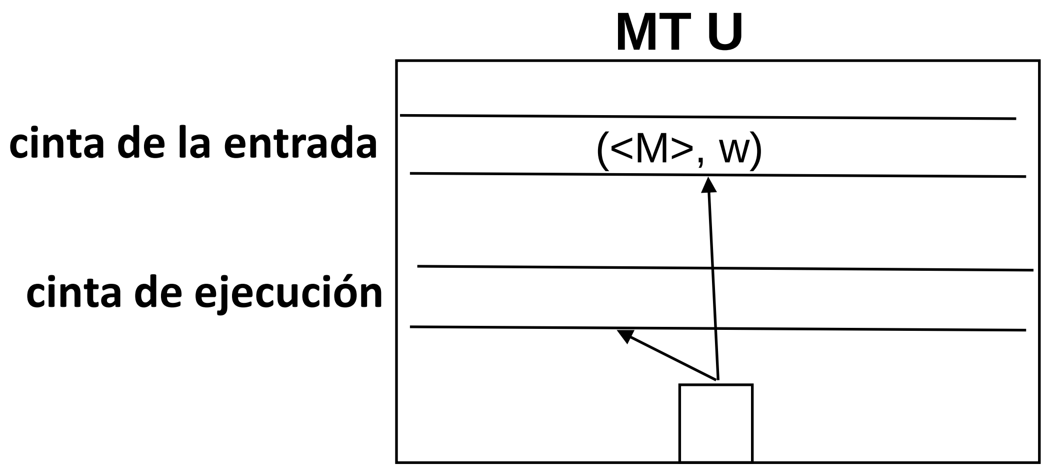
- Una máquina de Turing universal (MT U) es una máquina de Turing capaz de ejecutar otra (noción de **programa almacenado**, Turing 1936). El esquema más general es:





- La MT U recibe como entrada una **MT M** (codificada mediante una cadena <M>) y una **cadena w**, y **ejecuta M a partir de w**. Puede tener una o más cintas de ejecución.

Codificación de una máquina de Turing



- **Ejemplo:**

$M = (Q, \Sigma, \delta, q_1, q_A, q_R)$, tal que:

$Q = \{q_1, q_8\}$

$\Sigma = \{a, b, c\}$

$\delta = \{(q_1, a, q_8, c, R), (q_8, b, q_A, c, S), (q_1, b, q_R, b, S)\}$

Un estado q_i se codifica con el número i, y los estados q_A y q_R con Y y N, respectivamente:

Queda: **<M> = {(1,a,8,c,R), (8,b,Y,c,S), (1,b,N,b,S)}**

Y si la cadena w es aabb, la codificación completa del par (<M>, w) es:

(<**M>, aabb) = ({(1,a,8,c,R), (8,b,Y,c,S), (1,b,N,b,S)},aabb)**

Enumeración de las máquinas de Turing

- Para numerar las cadenas utilizaremos **el orden canónico:**

  1) De menor a mayor longitud.
  2) Con longitudes iguales desempata el orden alfanumérico (números, letras y símbolos especiales).

- Por ejemplo, las primeras cadenas según el orden canónico son:

λ																			
0	1	2	…	a	b	c	…	+	-	*	…								
00	01	02	…	0a	0b	0c	…	0+	0-	0*	…	10	11	12	…	1a	1b	1c	…
000	001	002	…	00a	00b	00c	…	00+	00-	00*	…	100	101	102	…	10a	10b	10c	…
etc.																			

- La siguiente MT M obtiene la MT M_i según el orden canónico. Dada una entrada i, M hace:

  1. Hace n := 0.
  2. Genera la siguiente cadena v según el orden canónico.
  3. Si v no es el código de una MT vuelve al paso 2.
  4. Si n = i, devuelve v **(v es el código de la MT M_i)** y para.
     Si n ≠ i, hace n := n + 1 y vuelve al paso 2.

  Validar que v es el código de una MT implica chequear que las 5-tuplas tengan la forma apropiada con números (estados), símbolos y las letras Y, N, L, R, S; que no haya 5-tuplas incompletas; etc.

Prueba de que R ⊂ RE ⊂ $\mathfrak{L}$

- Supongamos que la tabla T representa el comportamiento de **todas las MT** con respecto a **todas las cadenas**:

orden canónico (columnas) / orden canónico (filas)

T	w_0	w_1	w_2	w_3	w_4	…..
M_0	1	0	1	1	1	…..
M_1	1	0	0	1	0	…..
M_2	0	0	1	0	1	…..
M_3	0	1	1	1	1	…..
M_4	0	1	1	1	0	…..
…..	…..	…..	…..	….	…..	…..

$T(M_i, w_j) = 1$ significa que M_i acepta w_j, y $T(M_i, w_j) = 0$ significa que M_i rechaza w_j

- Cada fila de T representa **un lenguaje de RE:**

$L(M_0) = \{w_0, w_2, w_3, w_4. \ldots\}$
$L(M_1) = \{w_0, w_3, \ldots\}$
$L(M_2) = \{w_2, w_4. \ldots\}$
etc.

¿por qué están en RE?

- Por lo tanto, el conjunto de las filas de T representa **el conjunto RE.**

- Sea la diagonal de la tabla T:

T	w_0	w_1	w_2	w_3	w_4	.....
M_0	**1**	0	1	1	1	.....
M_1	1	**0**	0	1	0	.....
M_2	0	0	**1**	0	1	.....
M_3	0	1	1	**1**	1	.....
M_4	0	1	1	1	**0**	.....
.....	.....	.....	.....	....	.....	.....

- La diagonal de T representa otro lenguaje: **$D = \{w_i \mid M_i$ acepta $w_i\}$.**

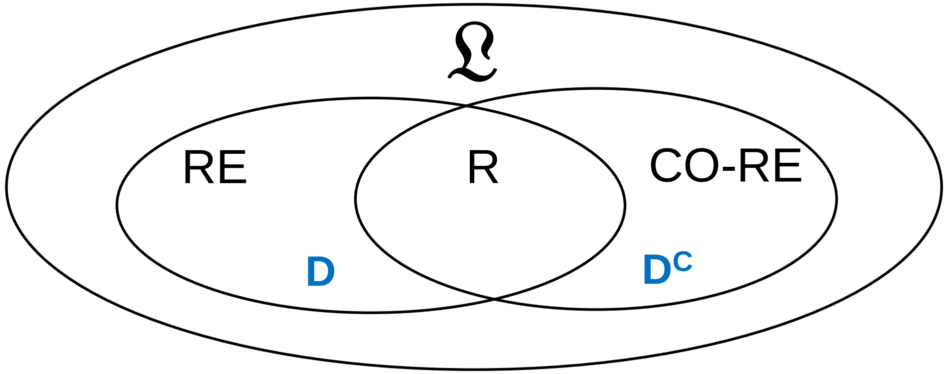
  Según el ejemplo: $D = \{w_0, w_2, w_3, \ldots\}$.

- Y la diagonal de T con los 1 y 0 invertidos este otro: **$D^C = \{w_i \mid M_i$ rechaza $w_i\}$.**

  Según el ejemplo: $D^C = \{w_1, w_4, \ldots\}$.

- Vamos a probar: (1) **D está en RE**
  (2) **D^C no está en RE**

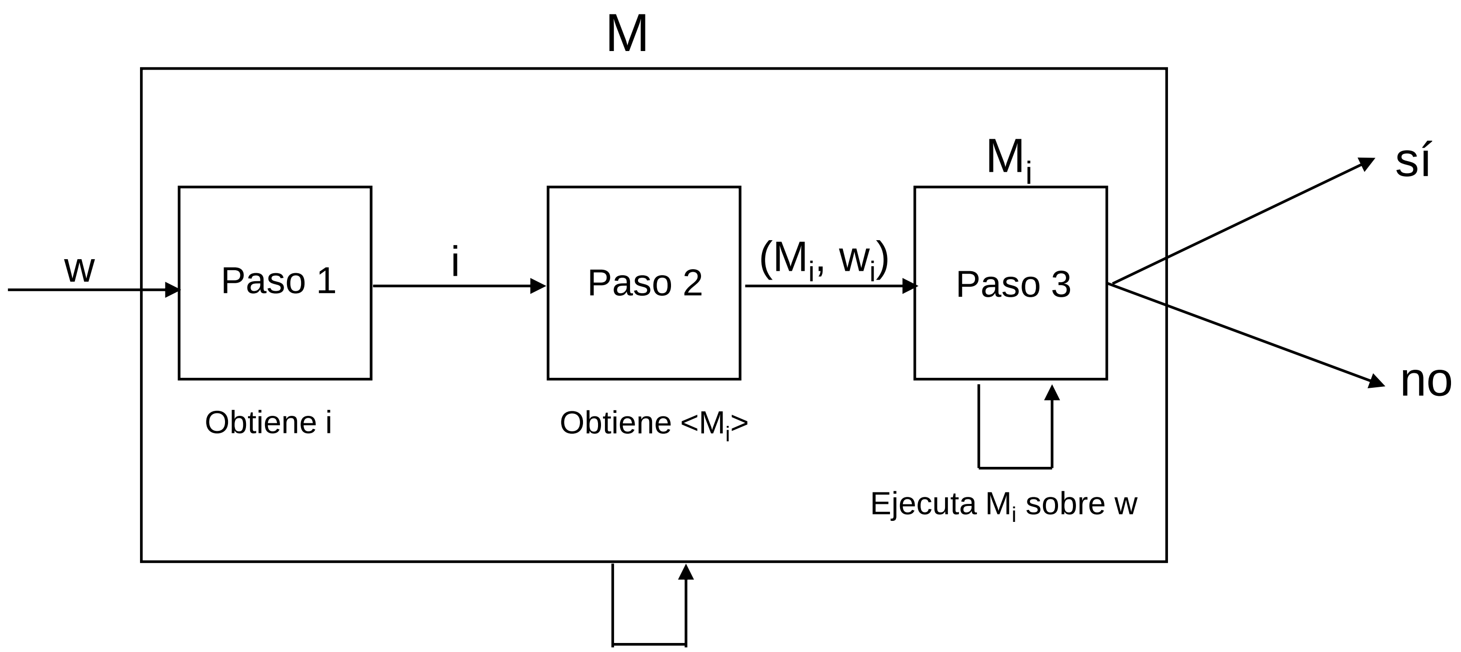
  y por lo tanto: (3) **D no está en R** ¿por qué?



(1) Primero probamos que D = $\{w_i \mid M_i$ acepta $w_i\}$ está en RE:

Vamos a construir una MT M que acepta el lenguaje D. Dada una entrada w, M hace:

1. Encuentra i tal que $w = w_i$ en el orden canónico (genera cadenas en orden canónico hasta encontrar w).
2. Encuentra el código <M_i> de la MT M_i (genera códigos de MT en orden canónico hasta llegar al i-ésimo).
3. Ejecuta M_i sobre w_i y acepta sii M_i acepta.



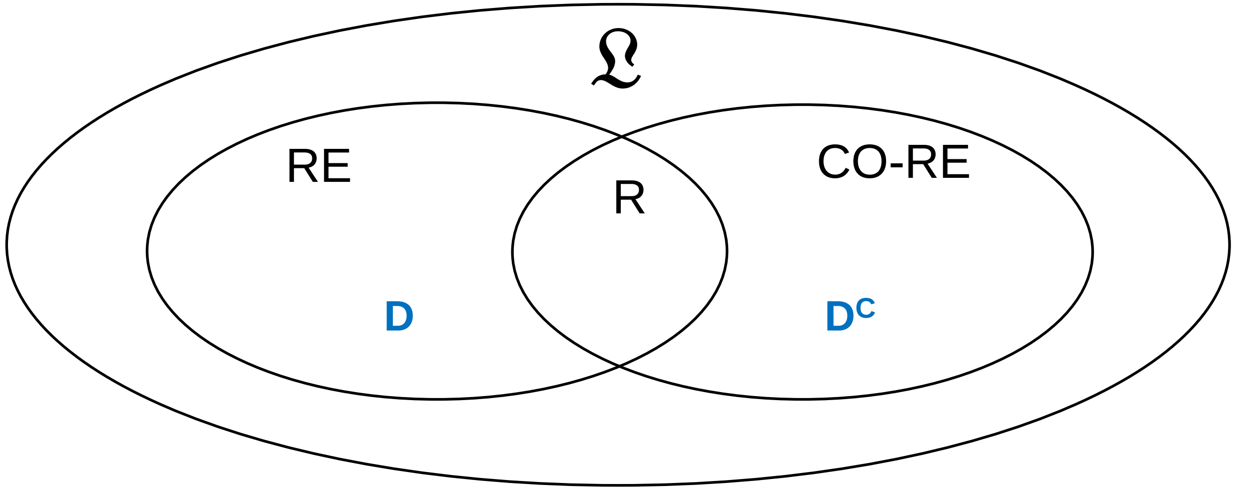
(2) **Ahora probamos que $D^C = \{w_i \mid M_i$ rechaza $w_i\}$ no está en RE**:

	T	w_0	w_1	w_2	w_3	w_4	…..
fila 0	M_0	**1**	0	1	1	1	…..
fila 1	M_1	1	**0**	0	1	0	…..
fila 2	M_2	0	0	**1**	0	1	…..
fila 3	M_3	0	1	1	**1**	1	…..
fila 4	M_4	0	1	1	1	**0**	…..
	…..	…..	…..	…..	….	…..	…..

- La diagonal d = (1,0,1,1,0,…) representa el lenguaje $D = \{w_i \mid M_i$ acepta $w_i\}$.
- **La diagonal invertida d^C = (0,1,0,0,1,…) representa el lenguaje $D^C = \{w_i \mid M_i$ rechaza $w_i\}$.**
- La fila 0 = (1,0,1,1,1,…) representa el lenguaje $L(M_0)$ y difiere de d^C en el 1er elemento.
- La fila 1 = (1,0,0,1,0,…) representa el lenguaje $L(M_1)$ y difiere de d^C en el 2do elemento.
- La fila 2 = (0,0,1,0,1,…) representa el lenguaje $L(M_2)$ y difiere de d^C en el 3er elemento.
- Etc.
- Así, todas las filas difieren de d^C. O lo mismo: **todos los lenguajes de RE difieren del lenguaje D^C.**
- Por lo tanto: **$D^C = \{w_i \mid M_i$ rechaza $w_i\}$ no está en RE.**

**Hemos utilizado la técnica de diagonalización.
En este caso no servía construir una MT.**

- **CONCLUSIÓN:** Hemos encontrado dos primeros lenguajes fuera de R:



A partir de estos primeros lenguajes, encontrados por **diagonalización**, se puede poblar la jerarquía de la computabilidad con una técnica más sencilla: **la reducción** (próxima clase).

$D = \{w_i \mid M_i \text{ acepta } w_i\}$ y $D^C = \{w_i \mid M_i \text{ rechaza } w_i\}$

- Otros lenguajes fuera de R que analizaremos oportunamente:

HP ={(<M>, w) | M para a partir de w}. Problema: ¿Acaso la MT M para a partir de la entrada w? **En RE – R.**

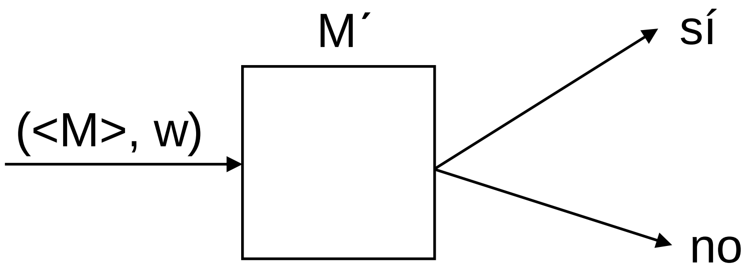
L_U = {(<M>, w) | M acepta w}. Problema: ¿Acaso la MT M acepta la entrada w? **En RE – R.**

$L_\emptyset$ = {<M> | L(M) = ∅}. Problema: ¿Acaso la MT M no acepta ninguna entrada? **En CO-RE – R.**

L_{Σ^*} = {<M> | L(M) = Σ*}). Problema: ¿Acaso la MT M acepta todas las entradas? **En 𝔏 – (RE ∪ CO-RE).**

Problema de la detención de una MT (*halting problem*)

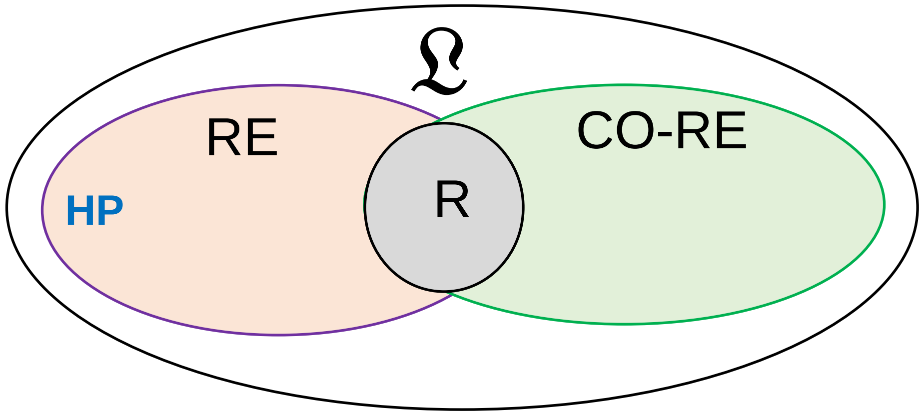
- Dada una MT M y una cadena w, ¿M para a partir de w?
- El lenguaje que representa el problema es **HP = {(<M>, w) | M para a partir de w}.**
- HP pertenece al conjunto **RE – R**. Fue el primer lenguaje encontrado fuera de R (Turing, 1936).
- Es de los más difíciles de RE. **Si HP estuviera en R, todo lenguaje de RE estaría en R (sería RE = R):**



M’ primero decide si M para a partir de w.

Si detecta que M no para, M´ rechaza.

Si detecta que M para, M´ ejecuta M desde w y responde como M.



HP está en la frontera de RE,
lo más lejos posible de R.
Se dice que es **RE-completo,**
identifica el grado de dificultad de RE.
Representa el **problema general de la indecibilidad**.

- HP está muy ligado a las **matemáticas.** Por ejemplo:

Conjetura de Goldbach
Todo número par mayor que 2 es la suma de 2 números primos:
4 = 2 + 2,   6 = 3 + 3,   8 = 3 + 5,   10 = 5 + 5,   12 = 5 + 7,   14 = 7 + 7,   etc.

Al día de hoy, la conjetura no ha sido probada.

Si HP fuera recursivo, la conjetura se resolvería fácilmente:
Se construye una MT M que vaya probando con 4, 6, 8, 10, etc.
Si M detecta un par que no sea suma de dos primos, para.
Así, si M_{HP} detecta que M para (no para), la Conjetura de Goldbach no vale (vale).

Ultimo Teorema de Fermat
Dados x, y, z, n, enteros, no existe n > 2 que cumpla $x^n + y^n = z^n$ (salvo las soluciones triviales con 1 y 0).

El teorema fue planteado en el siglo XVII y demostrado recién en 1995.

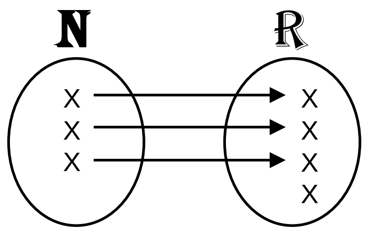
Si HP fuera recursivo, el teorema se hubiera resuelto enseguida:
Se construye una MT M que vaya probando con las distintas combinaciones de x, y, z, n.
Si M detecta una combinación que cumple la igualdad, para.
Así, si M_{HP} detecta que M para (no para), el Ultimo Teorema de Fermat no vale (vale).

Anexo de la clase teórica 3

Jerarquía de la computabilidad
(segunda parte)

Otra aplicación de la técnica de diagonalización

- Se puede probar por diagonalización (G. Cantor) que el conjunto ℝ de los números reales es más grande que el conjunto ℕ de los números naturales, es decir: **|ℝ| > |ℕ|.**



Hay más reales que naturales

Supongamos que |ℝ| = |ℕ|. Entonces, podemos enumerar los números reales. En particular los del intervalo (0, 1). Por ejemplo:

0,**1**287…
0,8**5**50…
0,13**8**0…
0,275**1**…
etc.

Sea el siguiente número real del intervalo (0, 1), fabricado considerando la diagonal pintada en azul:

- A los decimales 1 los cambiamos por el 2, y a los decimales distintos de 1 los cambiamos por el 1.
- Así obtenemos el número 0,**2112**…, distinto de todos los de la enumeración: difiere del 1ero en el 1er decimal, del 2do en el 2do decimal, etc.

Por lo tanto, no es posible enumerar los números reales, y así concluimos que **|ℝ| > |N|.**

Otra manera de probar RE $\subset \mathfrak{L}$

$|\Sigma^*|$ denota el tamaño de Σ^*, es decir, es la cantidad de todas las cadenas.

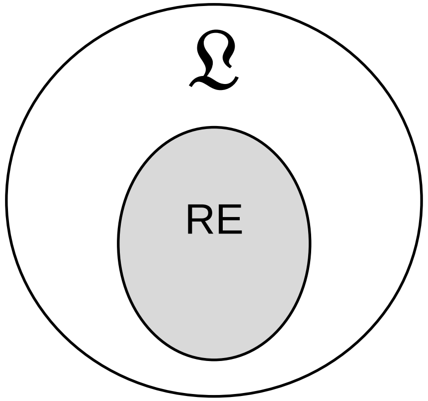
Por lo tanto, hay a lo sumo $|\Sigma^*|$ máquinas de Turing ¿por qué?

Por lo tanto, **RE tiene a lo sumo $|\Sigma^*|$ lenguajes** ¿por qué?

$\mathbb{P}(\Sigma^*)$ denota el conjunto de partes de Σ^*, es decir, el conjunto de todos los subconjuntos de Σ^*.

Por lo tanto, $|\mathbb{P}(\Sigma^*)|$ es la cantidad de lenguajes existentes, es decir, **$\mathfrak{L}$ tiene $|\mathbb{P}(\Sigma^*)|$ lenguajes.**

Teorema. Todo conjunto X, finito o infinito, cumple que $|X| < |\mathbb{P}(X)|$.



Hay más problemas que máquinas de Turing

De esta manera, como $|\Sigma^*| < |\mathbb{P}(\Sigma^*)|$, entonces **$|\text{RE}| < |\mathfrak{L}|$.**

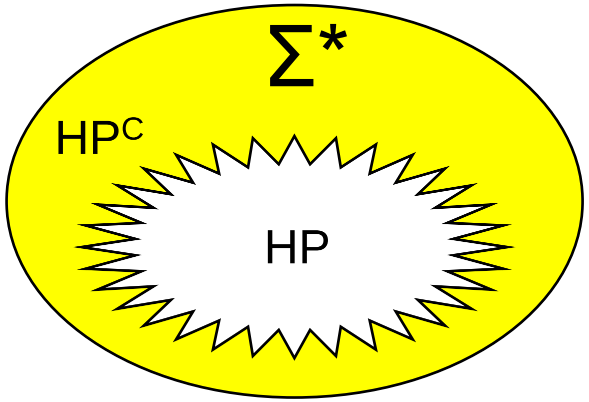
Computabilidad y tamaño de un lenguaje

- **HP $\subseteq \Sigma^*$, y HP es más difícil que Σ^*:**

  HP $\notin$ R y $\Sigma^* \in$ R ¿por qué $\Sigma^* \in$ R?

- **HP$^C \subseteq \Sigma^*$, y HP^C es más difícil que Σ^*:**

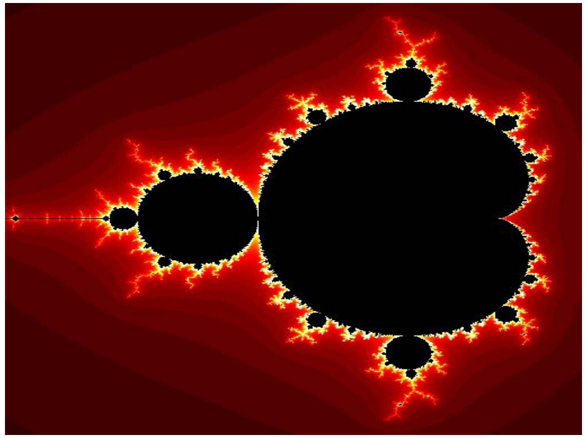
  HP$^C \notin$ RE y $\Sigma^* \in$ RE



- La computabilidad de un lenguaje (o problema) tiene más que ver con su definición, su **contorno** (representación gráfica), que con su tamaño.

- El contorno del Conjunto de Mandelbrot es un muy buen ejemplo de un lenguaje no recursivo.

CONJUNTO DE MANDELBROT



Clase práctica 3

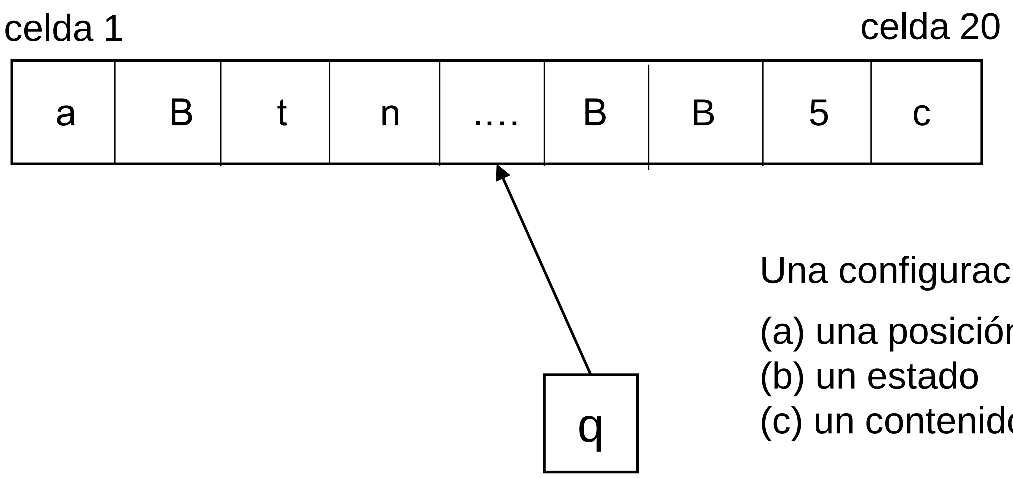
Jerarquía de la computabilidad
(segunda parte)

Cómo burlar al *halting problem*

- **Ejemplo 1.** Si una MT se mueve en un espacio limitado, se puede detectar cuándo entra en un loop.

  Por ejemplo, supongamos que una MT M con una cinta se mueve en no más de 20 celdas.

  ¿Por cuántas configuraciones distintas puede pasar M antes de loopear?



Una configuración de una MT tiene:

(a) una posición
(b) un estado
(c) un contenido

Si M tiene |Q| estados y |Σ| símbolos, antes de repetir una configuración hará a lo sumo:

$20.|Q|.|\Sigma|^{20}$ pasos (20 posiciones, |Q| estados, $|\Sigma|^{20}$ contenidos).

Se puede detectar si una MT loopea **ejecutándola y llevando la cuenta de sus pasos**.

Cómo burlar al *halting problem* (continuación)

- **Ejemplo 2.** ¿Cómo detectar si una MT M acepta al menos una cadena?

  Tenemos que tener cuidado en cómo construimos una MT M´ que chequee si M acepta una cadena.
  No sirve que M´ ejecute M sobre la 1ra cadena, luego sobre la 2da, luego sobre 3ra, …, porque M puede no detenerse en algunos casos.

Solución:

1. Hacer i := 1.
2. Ejecutar i pasos de M sobre todas las cadenas de longitud a lo sumo i.
3. Si M acepta alguna vez, aceptar.
4. Si no, hacer i := i + 1 y volver al paso 2.

pasos	**cadenas**
1	λ w_0 w_1 w_2 w_3 …
2	λ w_0 w_1 w_2 … w_0w_0 w_0w_1 w_0w_2 …
3	λ w_0 w_1 … w_0w_0 w_0w_1 … $w_0w_0w_0$ $w_0w_0w_1$ …
...	………………………………………………………..

Por ejemplo, si la primera cadena que M acepta mide 80 símbolos, y M la acepta en 120 pasos, entonces cuando la MT M´ ejecute 120 pasos de M sobre todas las cadenas de a lo sumo 120 símbolos la va a encontrar.

Prueba de que HP no es recursivo (por diagonalización, Turing 1936)

Supongamos que existe una MT M_{HP} que decide HP. **Llegaremos a una contradicción**.

Construimos una MT P de la siguiente manera. Dada una entrada <Q>, la MT P hace:

1. Ejecuta M_{HP} sobre (<Q>, <Q>).
2. Si M_{HP} responde que sí, entonces P "se hace entrar en un loop".
3. Si M_{HP} responde que no, entonces P para (responde indistintamente sí o no).

Es decir, P "le lleva la contra" a M_{HP} .

Veamos qué sucede cuando la entrada de P es su propio código <P> :

- Si P para desde <P>, significa que M_{HP} respondió no desde (<P>, <P>), es decir que P no para desde <P> .
- Si P no para desde <P>, significa que M_{HP} respondió sí desde (<P>, <P>), es decir que P para desde <P> .

En ambos casos obtuvimos una contradicción. Por lo tanto, no puede existir P. Y como P se construyó a partir de M_{HP}, entonces tampoco puede existir M_{HP} .

Es decir, **HP no es recursivo.**