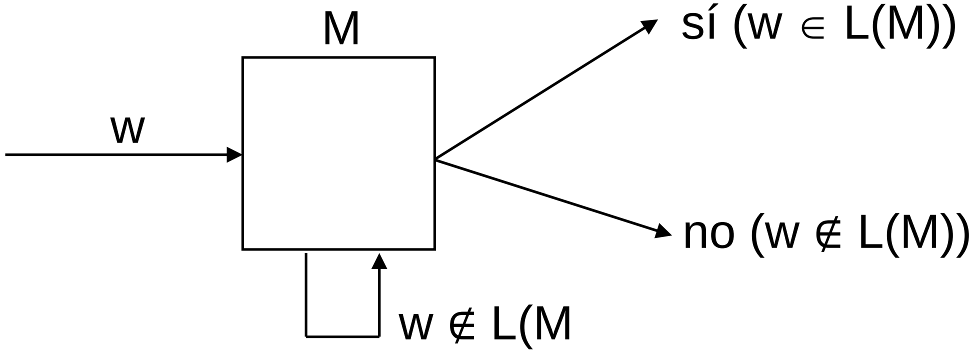
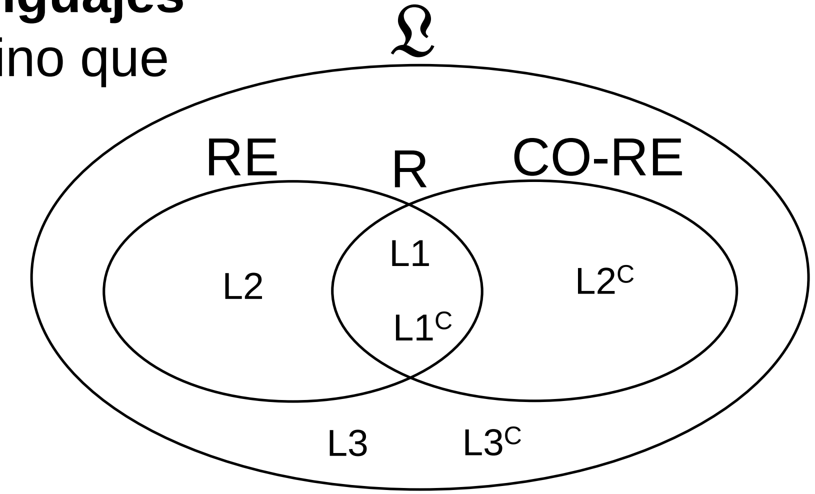
Clase 3. Problemas Indecidibles.

- En esta clase vamos a completar la prueba de la jerarquía de la computabilidad. Habíamos probado que **R = RE ∩ CO-RE** y adelantado **cómo se disponían los lenguajes en el mapa de la computabilidad** (ver el gráfico ejemplo).

- Ahora probaremos que efectivamente **R $\subset$ RE**, es decir que **hay lenguajes para los que no existen MT que los aceptan y paran siempre**, sino que para algunos casos negativos las MT loopean:

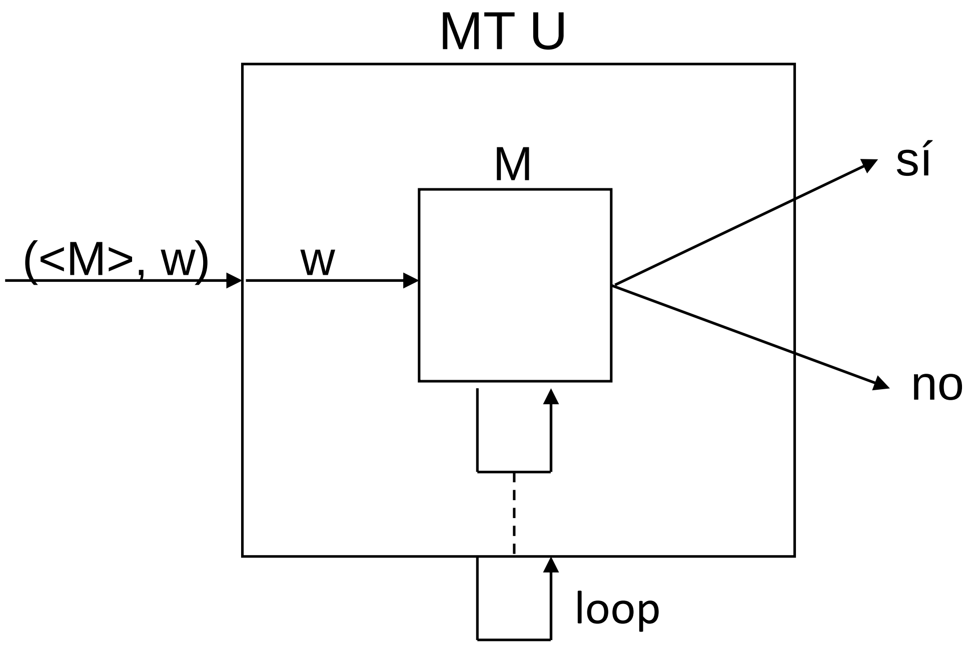


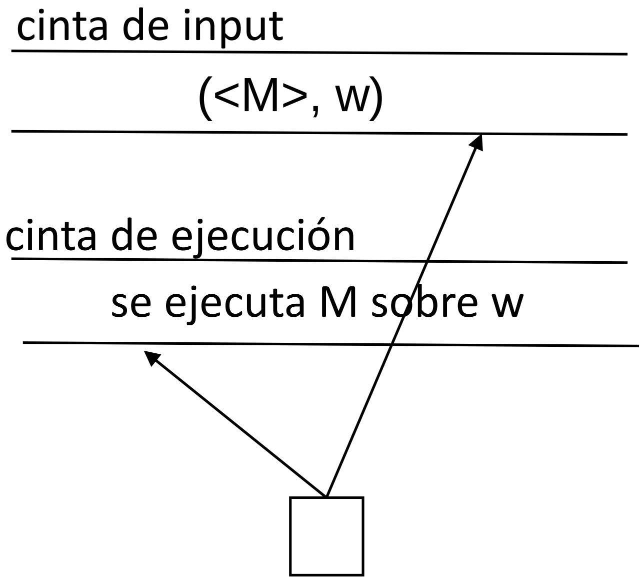


- También ya dijimos que probando **R $\subset$ RE** probaremos **RE $\subset \mathfrak{L}$ (hay lenguajes para los que no existen siquiera MT que los aceptan).** Ver el gráfico ejemplo: si L2 $\in$ RE y L2 $\notin$ R, entonces se cumple que $L2^C \notin$ RE, porque si L2 $\in$ RE y $L2^C \in$ RE, entonces L2 $\in$ R (R = RE ∩ CO-RE).
- Luego iremos poblando las distintas clases de la jerarquía con lenguajes particulares.

- Para lograr el objetivo debemos introducir primero la **Máquina de Turing Universal.**

- En lugar de construir una MT para cada lenguaje (o problema), la idea es tener una única MT capaz de ejecutar cualquier MT a partir de cualquier input (noción de **programa almacenado**, Turing 1936).





- La MT universal U recibe como input **una MT M** (codificada mediante una cadena que llamaremos <M>), y **una cadena w** (también codificada, pero para simplificar la notación no usaremos <w> sino directamente w), y **ejecuta M sobre w.**

Codificación de una MT

Dada $M = (Q, \Sigma, \Gamma, \delta, q_1, q_A, q_R)$, una manera estándar para la codificación <M> es la siguiente:

- **Q** $= \{q_1, q_2, \ldots, q_k\}$ se codifica con los números 1, 2, …, k, en notación binaria.
- **Σ** $= \{a_{i1}, a_{i2}, \ldots, a_{in}\}$ se codifica con los números $i_1, i_2, \ldots, i_n$, en notación binaria.
- **Γ** $= \{a_{i1}, a_{i2}, \ldots, a_{in}, \ldots, a_{im}\}$ se codifica con los números $i_1, i_2, \ldots, i_n, \ldots, i_m$, en notación binaria.
- El estado inicial, que por conveniencia llamaremos $\mathbf{q_1}$, se codifica con el número 1 en binario.
- Siendo k el tamaño de Q, es decir $|Q| = k$, $\mathbf{q_A}$ y $\mathbf{q_R}$ se codifican en binario con k+1 y k+2, resp.
- Finalmente, los movimientos $d \in$ **{L, R, S}** se codifican en binario con 1, 2, 3, resp.

El código <M> en definitiva es así: empieza con |Q|# y sigue con todas las 5-tuplas de δ, las cuales se separan entre sí con el símbolo #, al tiempo que sus componentes se separan con una coma.
No es necesario explicitar Q ni Σ ni Γ, porque se infieren de la codificación de δ.
Al comienzo se pone |Q| para que se puedan identificar los estados finales q_A y q_R.
De manera similar se puede codificar una MT con K cintas, con K pistas por cinta, etc. (ejercicio)

Por otro lado, **el código de** $\mathbf{w = w_1 w_2 \ldots w_n}$ se forma con los códigos de los w_i separados por comas.

Ejemplo de codificación de una MT

Sea una MT M tal que:

$Q = \{q_1, q_2\}$, $\Sigma = \{a_3, a_5\}$, $\Gamma = \{a_3, a_5, a_6\}$, el estado inicial es q_1, y:

$\delta(q_1, a_3) = (q_2, a_6, R)$
$\delta(q_2, a_5) = (q_A, a_6, S)$
$\delta(q_1, a_5) = (q_R, a_5, S)$

entonces queda:

<M> = 10#1,11,10,110,10#10,101,11,110,11#1,101,100,101,11

- La MT universal U tiene obviamente su propia función de transición δ_U. Dicha función debe interpretar adecuadamente los símbolos del input <M>. Dados en un momento dado q_i y a_k de la MT M ejecutada, δ_U debe recorrer <M> hasta encontrar, eventualmente, una tupla $(q_i, a_k, \ldots)$, y procesarla apropiadamente.

- Finalmente: los componentes del par (<M>, w) que recibe como input la MT U se van a separar con ##. P.ej. si $w = a_3a_3$, entonces el input de la MT U será: **<M>##11,11.**

Notar que se pueden enumerar las MT. La siguiente MT genera la MT i-ésima M_i, dado el número i:

1. Hacer n := 0.
2. Crear la siguiente cadena v de símbolos de {0 , 1 , # , **,** } según un orden canónico (ver abajo).
3. Si v no es el código de una MT (ver abajo), volver al paso 2.
4. Si n = i, aceptar (v es el código de la MT M_i). Si no, hacer n := n + 1 y volver al paso 2.

- **Validar que una cadena de símbolos v es el código de una MT** implica chequear que v tiene la forma |Q|#δ tal como se indicó antes: los números que representan estados no pueden superar el valor |Q|+2, los números que representan movimientos no pueden superar el valor 3, etc.
- **El orden canónico que utilizaremos es el lexicográfico** (por longitud, y con la misma longitud por orden alfanumérico). P.ej., asumiendo el orden 0 < 1 < **,** < #, las cadenas se sucederán así:

λ															
0	1	,	#												
00	01	0,	0#	10	11	1,	1#	,0	,1	,,	,#	#0	#1	#,	##
000	001	00,	00#	010	011	01,	01#	0,0	0,1	0,,	0,#	………		##,	###

……………………………………………………………………………...

- **De la misma forma se podrán enumerar los inputs w,** con $w = w_1w_2…w_n$. Por ejemplo, el input 1,1,10 estará antes que el input 1,1,11, y lo mismo el input 10 respecto del input 1,1.

- Con lo hecho antes, ahora sí podemos encontrar un **primer problema indecidible** (un primer lenguaje fuera de R):

- Usaremos la técnica de **diagonalización**, considerando una tabla T como la siguiente, que representa el comportamiento de **todas las MT** con respecto a **todos los inputs** posibles (los unos y ceros puestos son arbitrarios, se utilizan sólo para clarificar el concepto):

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	…..
…..	…..	…..	…..	….	…..	…..

- Las MT M_i y los inputs w_k se enumeran según el orden canónico que hemos definido.

- $T(M_i, w_k)$ es 1 o 0 según M_i acepta o rechaza w_k, respectivamente.

	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	…..
……	.....	.....	…..	.....	….	.....	…..

- Notar que la fila i de T representa el lenguaje, digamos L_i, aceptado por la MT M_i :

  $L(M_i) = L_i = \{w_k \mid M_i$ acepta $w_k\}$

- Por ejemplo: $L(M_0) = L_0 = \{w_0, w_2, w_3, w_4, \ldots\}$

  $L(M_1) = L_1 = \{w_0, w_3, \ldots\}$

  $L(M_2) = L_2 = \{w_2, w_4, \ldots\}$

  Etc.

- Como las M_i son **todas** las MT, entonces **los L_i constituyen en su conjunto la clase RE completa. Es decir, RE = $\{L(M_0), L(M_1), L(M_2), L(M_3), L(M_4), \ldots\} = \{L_0, L_1, L_2, L_3, L_4, \ldots\}$.**

- Consideremos ahora 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, es decir (1, 0, 1, 1, 0, …), también representa un lenguaje, digamos D:

$$\mathbf{D = \{w_i \mid M_i \text{ acepta } w_i\}}$$

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

- Además, invirtiendo los unos y ceros de la diagonal (1, 0, 1, 1, 0, …), se obtiene (0, 1, 0, 0, 1, …), que representa el lenguaje complemento de D:

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

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

Principio de Diagonalización: en una tabla T de unos y ceros con diagonal d, si d^C resulta de invertir los unos y ceros de d, entonces todas las filas de T difieren de d^C. Veámoslo en la tabla ejemplo:

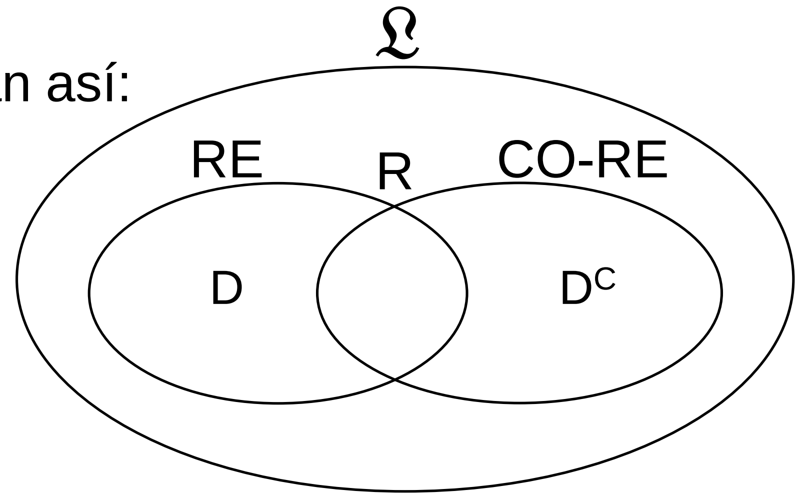
	T	w_0	w_1	w_2	w_3	w_4	…..
fila 0	M_0	**1**	0	1	1	1	…..
fila 1	M_1	1	**0**	0	1	0	…..
fila 2	M_2	0	0	**1**	0	1	…..
fila 3	M_3	0	1	1	**1**	1	…..
fila 4	M_4	0	1	1	1	**0**	…..
	…..	…..	…..	…..	….	…..	…..

- d = (1,0,1,1,0,…) y d^C = (0,1,0,0,1,…)
- La fila 0 = (1,0,1,1,1,…) difiere de d^C en el 1er elemento.
- La fila 1 = (1,0,0,1,0,…) difiere de d^C en el 2do elemento.
- La fila 2 = (0,0,1,0,1,…) difiere de d^C en el 3er elemento.
- Etc.

Pero entonces, **todos los lenguajes $L(M_i) = L_i = \{w_k \mid M_i$ acepta $w_k\}$, asociados a las filas i, difieren del lenguaje $D^C = \{w_i \mid M_i$ rechaza $w_i\}$, asociado a d^C**. Y como los lenguajes $L(M_i) = L_i$ constituyen en su conjunto la clase RE:

¡El lenguaje $D^C = \{w_i \mid M_i$ rechaza $w_i\}$ no pertenece a RE!

- Hemos encontrado un primer lenguaje fuera de RE, el lenguaje $D^C = \{w_i \mid M_i \text{ rechaza } w_i\}$, **y así hemos probado que RE $\subset \mathfrak{L}$.**
- Se prueba fácilmente que $D = \{w_i \mid M_i \text{ acepta } w_i\} \in$ RE. Enseguida lo veremos, ahora asumámoslo.
- Obviamente $D \notin R$, porque si $D \in R$ entonces también $D^C \in R$, y así $D^C \in$ RE (absurdo). Por lo tanto, hemos encontrado también un primer lenguaje de RE – R, el lenguaje $D = \{w_i \mid M_i \text{ acepta } w_i\}$, **y así hemos probado también que R $\subset$ RE.**
- Las ubicaciones de D y D^C quedan así:



- D^C es aún "más difícil" que D (ni siquiera es recursivamente numerable).

Veamos que $D = \{w_i \mid M_i$ acepta $w_i\} \in$ RE:

La siguiente MT M_D acepta el lenguaje D. Dado un input w, M_D hace:

1. Encuentra qué posición ocupa w en el orden canónico, es decir qué cadena w_i es (va generando cadenas, una a una, en el orden canónico, hasta encontrar w y así **detecta el índice i**).
2. Con el índice i como dato **genera el código $<M_i>$** (va generando códigos de MT en el orden canónico, uno a uno, hasta llegar al i-ésimo).
3. Finalmente **ejecuta M_i sobre w, y acepta si y sólo si M_i acepta w**.

De la misma forma podemos probar que 2 lenguajes similares a D están en RE:

$L_U = \{(<M>,w) \mid M$ acepta $w\}$. Es el **problema universal de aceptación**.

$HP = \{(<M>,w) \mid M$ para sobre $w\}$. Es el **problema de la parada (Halting Problem)**.

Queda como ejercicio probar que $L_U \in$ RE y HP $\in$ RE.

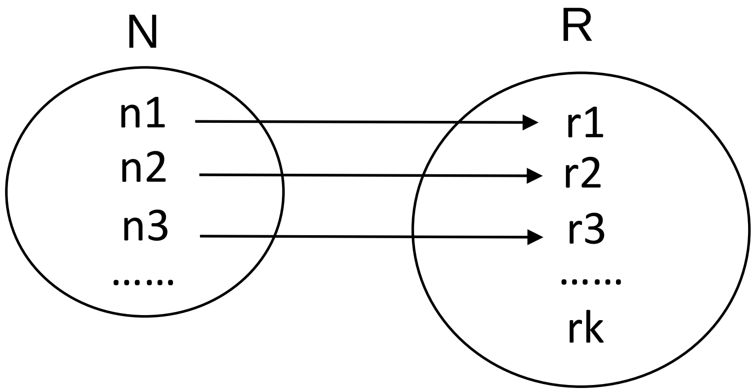
Los lenguajes L_U y HP no están en R. Volveremos a ellos enseguida.

- Antes de seguir con la jerarquía de la computabilidad y los problemas indecidibles, dediquemos un poco más de tiempo a la **diagonalización.**
- Georg Cantor creó el método para probar que |R| > |N|, es decir que **hay más números reales que números naturales**:
- Supongamos que |R| = |N|. En otras palabras, que podemos enumerar los números reales, y en particular los del intervalo (0, 1). Por ejemplo, sea esta enumeración:

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

Notar que el siguiente número real en el intervalo (0,1) no está en la enumeración: lo fabricaremos a partir de los decimales de la diagonal (en azul), tal que a los 1 los cambiaremos por 2 y a los distintos de 1 por 1. Queda: 0,**2112**…
Dicho número difiere de todos los de la enumeración. Difiere del 1ro en el 1er decimal, del 2do en el 2do decimal, del 3ro en el 3er decimal, y así con todos.
Por lo tanto, no se pueden enumerar los números reales, se cumple que **|R| > |N|.**

- Expresado de otra manera, **no se puede establecer una biyección entre R y N**:



Hay más reales que naturales, hay elementos r de R que no se corresponden con elementos n de N

- En cambio, sí se pueden enumerar los números enteros, los números racionales, las cadenas de Σ^* con $\Sigma = \{a_1, a_2, a_3, \ldots, a_k\}$, etc. Queda como ejercicio.

- El tamaño de N es el **primer cardinal infinito**, se lo conoce como $\boldsymbol{\aleph_0}$. El tamaño de R se conoce como **C** (por el **continuo**). La **Hipótesis del Continuo** enuncia que no hay un cardinal infinito intermedio entre $\aleph_0$ y C (establece que C es el 2do cardinal infinito, $\aleph_1$).

- Por otra parte, se prueba que **|P(N)| = |R|**, es decir que hay tantos subconjuntos de N como números reales. Por lo tanto, como |R| > |N|, entonces |P(N)| > |N| (la prueba no es complicada: tener en cuenta que a la derecha de la coma de un número real hay $10^{|N|}$ continuaciones decimales posibles).

- **Así las cosas, con |R| = |P(N)| > |N|, podemos probar de una manera alternativa que RE $\subset \mathfrak{L}$:**

- |RE| = |N|. Los lenguajes de la clase RE se pueden enumerar, porque las MT se pueden enumerar como ya vimos.

- |Σ*| = |N|. Lo dijimos también recién, se pueden enumerar las cadenas generadas a partir de un alfabeto Σ.

- Como la clase $\mathfrak{L}$ es el conjunto de todos los lenguajes, está formada por todos los subconjuntos posibles de cadenas de Σ*, es decir que |$\mathfrak{L}$| = |P(Σ*)| = |P(N)|

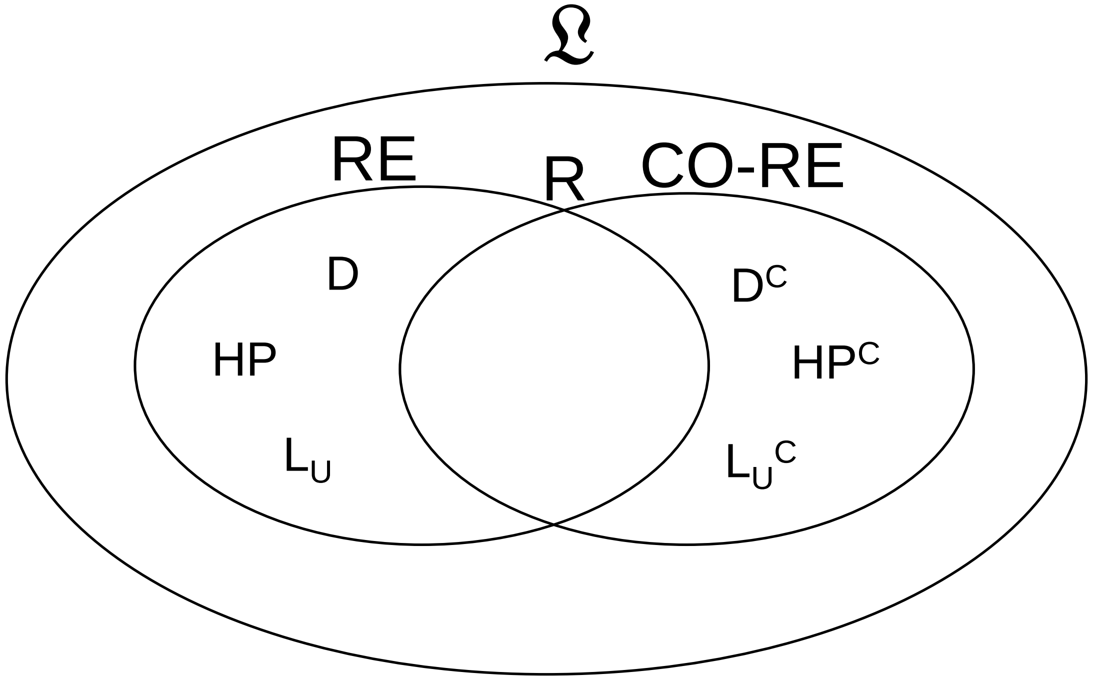
- Por lo tanto, **|RE| = |N| < |P(N)| = |$\mathfrak{L}$|**

Hay más lenguajes que lenguajes recursivamente numerables.

O lo que es lo mismo, hay más problemas que MT que los resuelven.

No todo es computable.

- **Volvamos a la jerarquía de la computabilidad**. Vimos que podemos poblarla así:



- La prueba de que **HP = {(<M>,w) | M para sobre w}** $\in$ (RE – R) también se puede hacer por **diagonalización**. La presentó Turing en su artículo de 1936. Es un poco más complicada que la que vimos para probar que D $\in$ (RE – R). Se muestra luego.

- Y la prueba de que **L_U = {(<M>,w) | M acepta w}** $\in$ (RE – R) la veremos la clase que viene con una técnica más sencilla, la **reducción de problemas**.

Problema de la Parada (Halting Problem)

- El lenguaje HP = {(<M>,w) | M para sobre w} es **de los más difíciles de RE** (se usa la expresión "HP es RE-completo", porque **representa la dificultad de la clase RE**).

- Que HP no sea recursivo significa que no existe un mecanismo **general** para decidir si una MT para a partir de un input (**es indecidible el problema de si un programa termina**).

- Esto no quiere decir que para una MT M y un input w **particulares**, nosotros no podamos determinar si M para sobre w. Efectivamente, para cada par (<M>,w), podemos resolver la cuestión agudizando el ingenio, analizando el código <M>, observando las características de w, etc. Que HP $\notin$ R significa que no existe un algoritmo, programa, MT, que **para todo input de la forma (<M>, w)** decida si M para o no a partir de w.

- Una manera de ver que HP representa la dificultad de la clase RE es probando:

  Si HP $\in$ R, entonces R = RE

- Es decir, si hubiera una MT que decide HP, también habría una MT que decide cualquier lenguaje de RE (absurdo). La prueba es la siguiente:

Partimos del absurdo de suponer que HP $\in$ R. Sea M_{HP} una MT que decide HP.

Y sea L algún lenguaje cualquiera de RE. Sea M_1 una MT que acepta L.

Veamos que también existe una MT M_2 que decide L (**y así L estará en R**). Dado un input w, la MT M_2 hace:

1. Ejecuta M_{HP} sobre el par (<M_1>,w).
2. Si M_{HP} rechaza, significa que M_1 no para a partir de w, y por lo tanto rechaza.
3. Si M_{HP} acepta, significa que M_1 para a partir de w. Así, ejecuta M_1 sobre w y acepta si y sólo si M_1 acepta.

Claramente M_2 acepta L y para siempre.

- El problema HP está muy ligado a las matemáticas. **Muchos problemas matemáticos serían de fácil resolución si HP fuera decidible**:

- **Conjetura de Goldbach:** "Todo número par mayor que 2 es la suma de 2 números primos".

  P.ej., 4 = 2 + 2, 6 = 3 + 3, 8 = 3 + 5, 10 = 5 + 5, 12 = 5 + 7, …

  Al día de hoy, **la conjetura no ha podido ser probada**. Si HP estuviera en R, es decir si tuviéramos una MT M_{HP} que decide HP, **habría una MT M_G para aceptar o rechazar la conjetura:**

  Construcción de M_G

  \- Sea M una MT que va obteniendo por cada número par p, los 2 primos que sumados dan p, y si para un caso de p **no encuentra los 2 primos**, entonces **para**.

  \- **M_G invoca a M_{HP} con input (<M>, w)** - no importa el w, M_{HP} no lo usa -. Si **M_{HP} acepta**, es decir si M para, entonces **M_G rechaza**. Y si **M_{HP} rechaza**, es decir si M no para, entonces **M_G acepta**.

- Otro ejemplo de lo anterior es el **Ultimo Teorema de Fermat**:

" Para cualesquiera enteros x, y, z distintos de 0, no hay un entero $n > 2$ que cumpla $x^n + y^n = z^n$ ".

Fermat en 1637 indicó que la prueba de este teorema no la presentaba "porque no tenía espacio en el margen de su libro". **El teorema se demostró en 1995 (Andrew Wiles).**

Si tuviéramos una MT M_{HP} que decide HP, **habría una MT M_F para demostrar el teorema (de un modo mucho más fácil que el utilizado por Wiles):**

Construcción de M_F

- Sea M una MT que va chequeando para toda tupla (x, y, z, n) con las hipótesis dadas, que no cumple la ecuación $x^n + y^n = z^n$. Si detecta una tupla que la **cumple**, entonces **para**.

- M_F invoca a M_{HP} con input (<M>, w) - no importa el w, M_{HP} no lo usa -. Si M_{HP} **acepta**, es decir si M para, entonces M_F **rechaza**. Y si M_{HP} **rechaza**, es decir si M no para, entonces M_F **acepta**.

- **La técnica de la diagonal de Cantor es muy efectiva, se basa en la autorreferencia.** Turing la utilizó para encontrar un primer ejemplo de lenguaje fuera de R, el lenguaje HP, el **Halting Problem.**

- Con la misma técnica Gödel probó la **incompletitud de la aritmética:** ”En cualquier axiomática consistente de la aritmética, hay enunciados verdaderos que no se pueden probar”. Dada una axiomática, Gödel encuentra un enunciado que establece que él mismo no puede ser demostrado.

- También con la misma técnica, Russell “derrumbó” la teoría inicial de conjuntos de Cantor (**Paradoja de Russell):** “¿El conjunto de todos los conjuntos que no pertenecen a sí mismos, pertenece a sí mismo?”. Si la respuesta es sí llegamos a que no, y viceversa. (Hay paradojas similares que se explican más fácil, como la **Paradoja del Barbero** y la **Paradoja del Mentiroso**.)

- En la clase siguiente aprenderemos una técnica más sencilla que la diagonalización para seguir poblando la jerarquía de la computabilidad, **la reducción de problemas.**

Sobre la computabilidad y el tamaño de un lenguaje

$\Sigma^* \in R$

$HP \notin R$

$HP^C \notin RE$

- Si $L_1 \subseteq L_2$ y $L_2 \in R$, ¿se cumple que $L_1 \in R$?

  No. Por ejemplo:

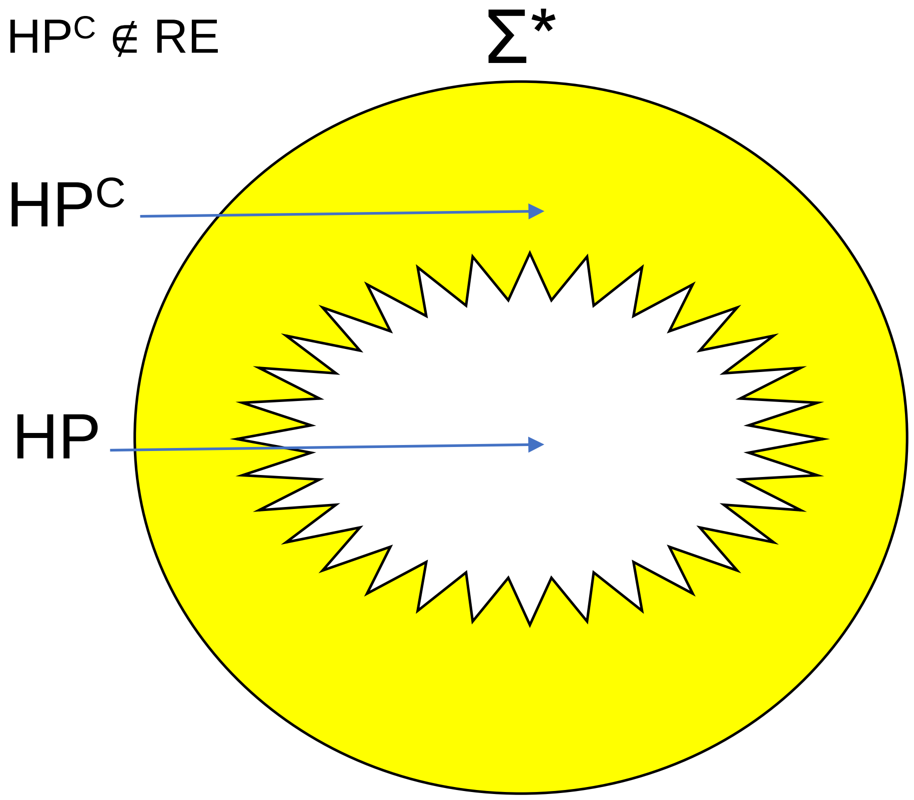
  $HP \subseteq \Sigma^*$, $\Sigma^* \in R$, pero se cumple $HP \notin R$.

- Y si $L_1 \subseteq L_2$ y $L_2 \in RE$, ¿se cumple que $L_1 \in RE$?

  Tampoco. Por ejemplo:

  $HP^C \subseteq \Sigma^*$, $\Sigma^* \in RE$, pero se cumple $HP^C \notin RE$.

- La computabilidad de un lenguaje o problema no tiene que ver con su **tamaño** o **densidad**, es decir con la cantidad de cadenas que contiene, como sí veremos que sucede en la complejidad computacional temporal. La computabilidad se relaciona con la **definibilidad**, con el **contorno** de los conjuntos involucrados.



Chequear si una cadena está en Σ^* es fácil (Σ^* es recursivo). Y aunque HP y HP^C están incluidos en Σ^*, chequear si una cadena está en ellos es difícil.

Ejemplo (una manera de “burlar” al Halting Problem).

Sea L_{20} = {<M> | M es una MT que a partir del input vacío λ nunca sale de las celdas 1 a 20}.
Vamos a probar que $L_{20} \in R$.

Si <M> $\in L_{20}$, entonces hay un número máximo C de configuraciones distintas por las que pasa M, con |Q| estados y |Γ| símbolos, antes de entrar en loop, con $C = 20.|Q|.|\Gamma|^{20}$.

Vamos a construir una MT M_{20} que acepta L_{20} y para siempre, usando C. La MT M_{20} tiene 5 cintas.
Dado un input w, M_{20} trabaja de la siguiente manera:

1. Si w no es un código <M> válido, rechazar.
2. Calcular y escribir C en la cinta 5. Hacer i := 1 en la cinta 3 (i guardará la posición del cabezal de la MT M que se va a ejecutar). Y hacer n := 0 en la cinta 4 (n guardará el número de pasos ejecutados de M).
3. Ejecutar el paso siguiente de M con input λ en la cinta 2.
4. Actualizar adecuadamente el valor de i en la cinta 3. Si i = 0 o 21, rechazar. Si M paró, aceptar.
5. Hacer n := n + 1 en la cinta 4. Si n = C, aceptar.
6. Volver al paso 3.

Queda como ejercicio probar que M_{20} para siempre y que $L(M_{20}) = L_{20}$.

Otro ejemplo para “burlar” al Halting Problem.

Sea $L = \{<M> \mid L(M) \neq \varnothing\}$.
El lenguaje representa el problema de determinar si una MT M acepta al menos un input.

Se cumple que $L \in RE$. Tenemos que tener cuidado en cómo construimos una MT M_L que lo acepte: no sirve ejecutar directamente M a partir de todas las cadenas generadas en el orden canónico, porque puede existir una cadena que M acepta pero que sea posterior a otra en la que M loopea.

La MT M_L propuesta trabaja de la siguiente manera a partir de un input w:

1. Si w no es un código <M> válido, rechazar.
2. Hacer i := 1.
3. Ejecutar M a lo sumo i pasos, a partir de todas las cadenas v tales que $|v| \leq i$. Si en algún caso M acepta, aceptar.
4. Hacer i := i + 1 y volver al paso 3.

Queda como ejercicio probar que $L(M_L) = L$.

Ejemplo. Prueba por diagonalización de que HP $\notin$ R (basada en la del artículo de Turing de 1936).

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

A partir de la MT M_{HP} construimos una MT P de la siguiente manera. Dado un input w, si w no es un código válido de MT, digamos <Q>, P rechaza, y en caso contrario:

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

Es decir, P es una MT que "le lleva la contra" a M_{HP} con respecto a inputs que son códigos de MT.

Veamos qué sucede cuando P se ejecuta sobre su propio código <P>:

- Si P **para** sobre <P>, significa que M_{HP} respondió que P **no para** sobre <P>.
- Y si P **no para** sobre <P>, significa que M_{HP} respondió que P **para** sobre <P>.

Por lo tanto, no puede existir una MT con las características de P. Y como P se construyó a partir de M_{HP}, **entonces tampoco puede exisitr una MT con las características de M_{HP}. HP no es recursivo.**