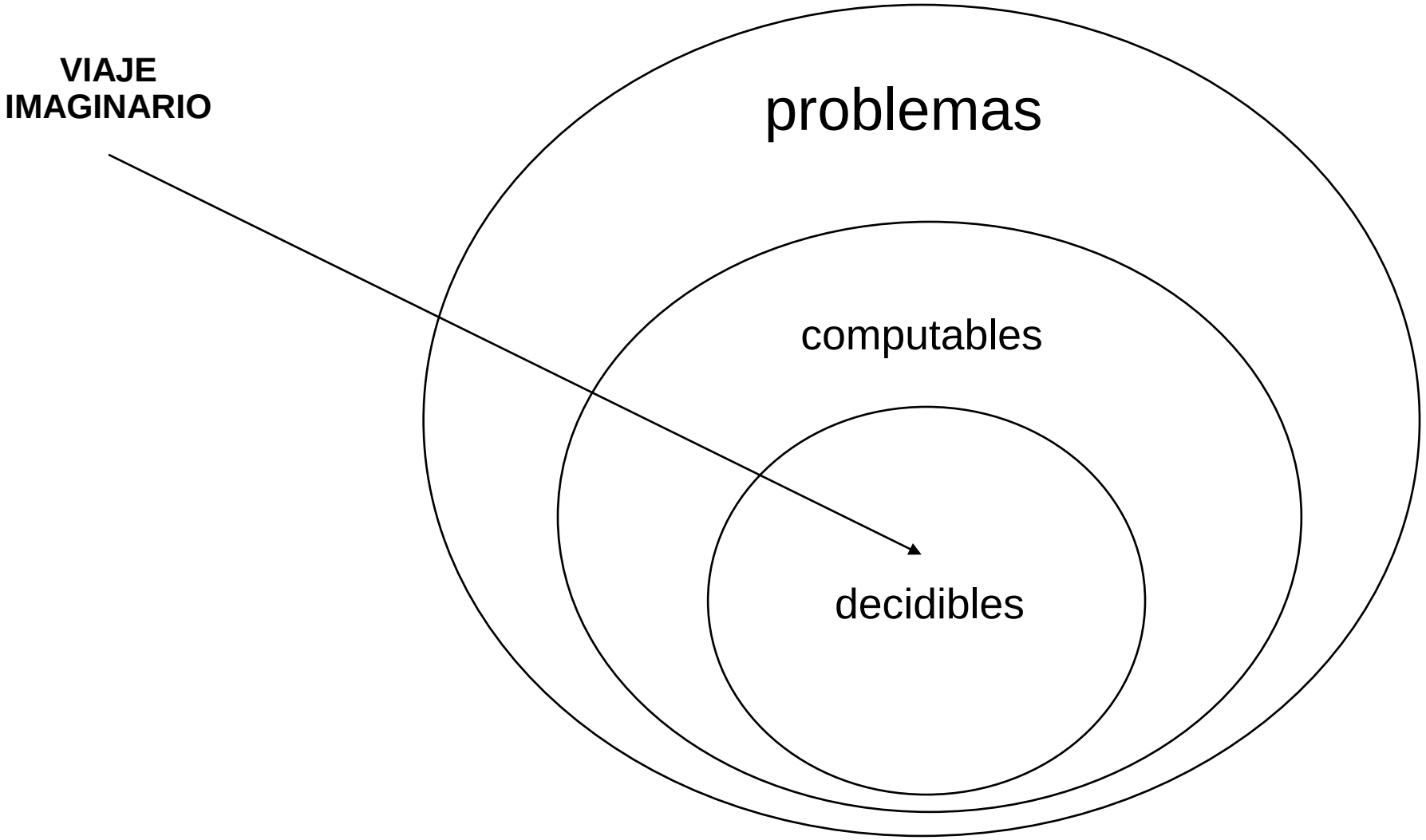
Clase teórica 2

Jerarquía de la computabilidad

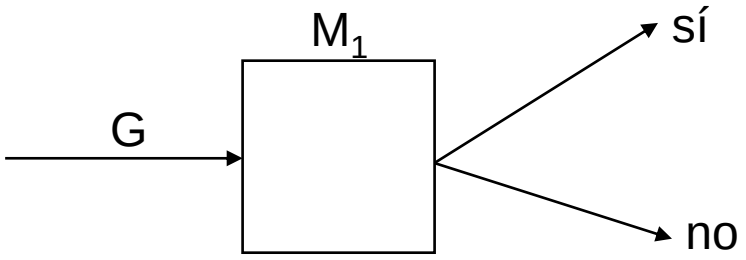
Iniciamos el viaje imaginario



- **De problemas de búsqueda a problemas de decisión (o problemas sí/no).**

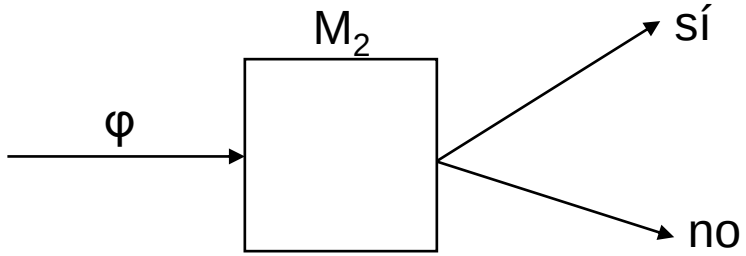
Los resolvemos con máquinas de Turing (MT) que aceptan **lenguajes: problemas "=" lenguajes.**

Ejemplos:



$L(M_1)$ = {G | G es un grafo que tiene un camino del vértice 1 al vértice n}.

$L(M_1)$ representa el problema de decisión: **"¿El grafo G tiene un camino del vértice 1 al vértice n?".**

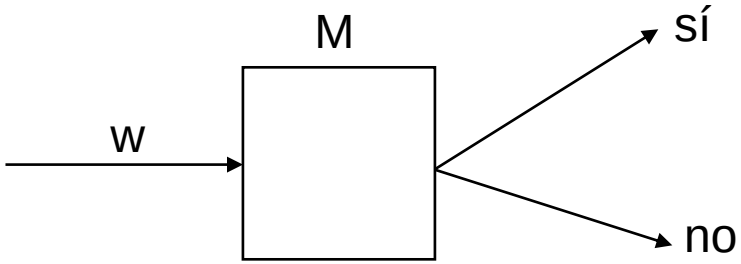


$L(M_2)$ = {φ | φ es una fórmula booleana satisfactible, es decir, existe una asignación de valores de verdad que la hace verdadera}.

$L(M_2)$ representa el problema de decisión: **"¿La fórmula booleana φ es satisfactible?".**

- **Tres posibilidades:**

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

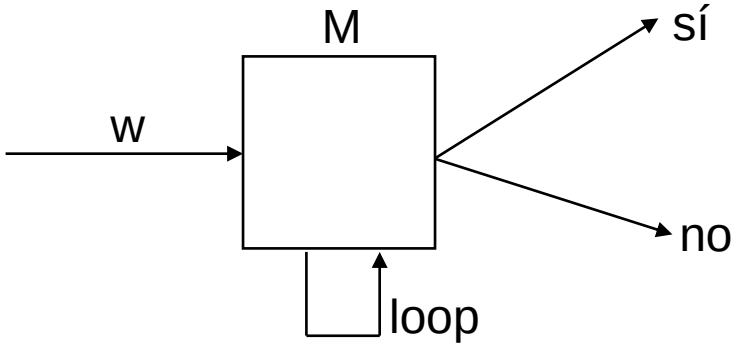


- Si w ∈ L(M), M acepta.
- Si w ∉ L(M), M rechaza.

Lenguajes **recursivos.**

Llamaremos **R** al conjunto de estos lenguajes.

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



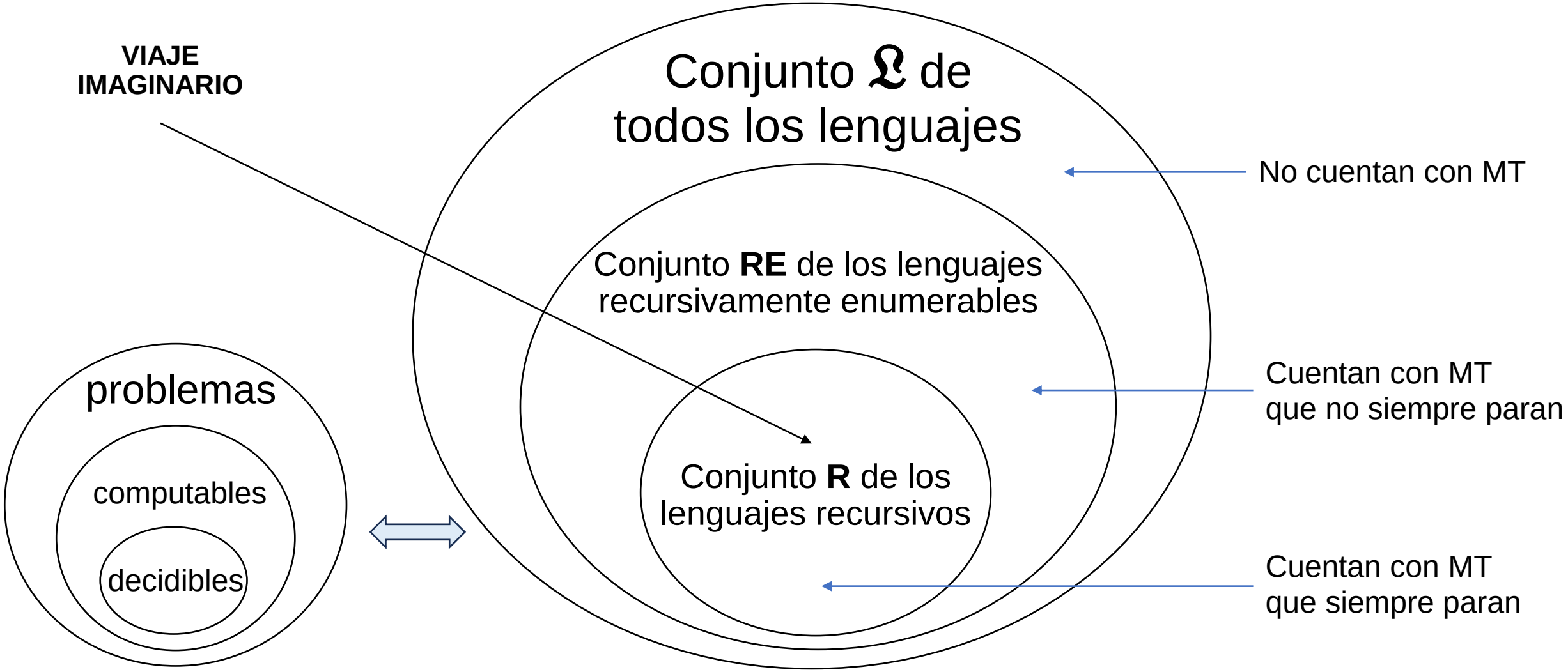
- Si w ∈ L(M), M acepta.
- Si w ∉ L(M), M rechaza o loopea.

Lenguajes **recursivamente enumerables**.

Llamaremos **RE** al conjunto de estos lenguajes.

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

Primera versión de la jerarquía de la computabilidad



- **Formalizando las definiciones:**

- Un lenguaje L es **recursivo** ($L \in R$) sii existe una MT M_L que lo **acepta y para siempre**.

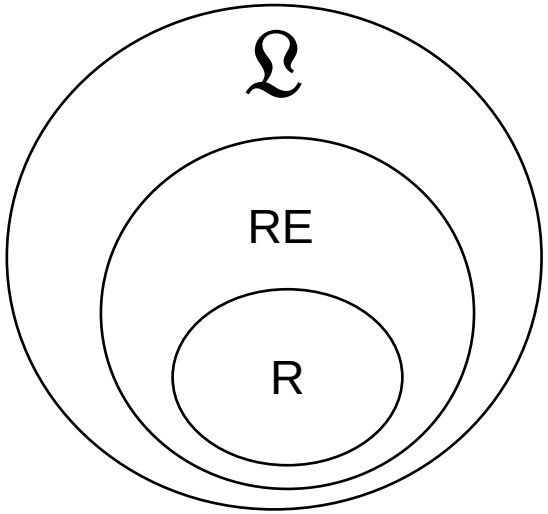
  Para toda cadena w del conjunto universal de cadenas Σ^*:

  - Si $w \in L$, entonces M_L a partir de w para en su estado q_A
  - Si $w \notin L$, entonces M_L a partir de w para en su estado q_R

- Un lenguaje L es **recursivamente numerable** ($L \in RE$) sii existe una MT M_L que lo **acepta**.

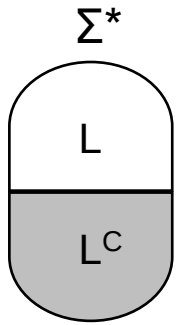
  Para toda cadena w del conjunto universal de cadenas Σ^*:

  - Si $w \in L$, entonces M_L a partir de w para en su estado q_A
  - Si $w \notin L$, entonces M_L a partir de w para en su estado q_R o no para



- Se cumple por definición que $\mathbf{R} \subseteq \mathbf{RE} \subseteq \Omega$ (ejercicio).
- También se cumple $\mathbf{R} \subset \mathbf{RE} \subset \Omega$ (lo probamos en la próxima clase).

Algunas propiedades de la clase R



Lema 1. **Si $L \in R$, entonces $L^C \in R$**, tal que L^C es el complemento de L.

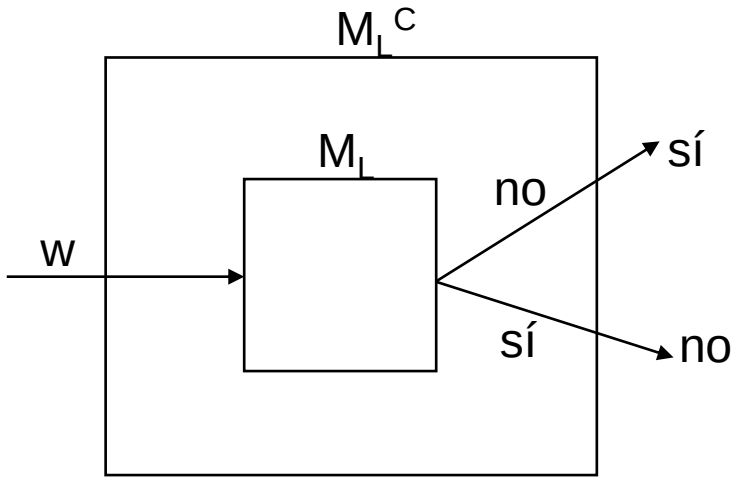
Definición: $L^C = (\Sigma^* - L)$, o en otras palabras, L^C tiene las cadenas que no tiene L.

Prueba.

1. Idea general.

Dada una MT M_L que acepta L y para siempre, la idea es construir una MT $M_L{}^C$ que acepte L^C y pare siempre.

Propuesta de solución: construir $M_L{}^C$ como M_L pero permutando sus estados finales:



M_L acepta L y para siempre por hipótesis ($L \in R$).

Entonces $M_L{}^C$ acepta L^C y para siempre,

y así también $L^C \in R$.

2. Construcción de la MT $M_L{}^C$.

Si: $M_L = (Q, \Sigma, \delta, q_0, q_A, q_R)$

entonces: $M_L{}^C = (Q, \Sigma, \delta', q_0, q_A, q_R)$

tal que δ y δ' son idénticas salvo que con los estados q_A y q_R **permutados.**

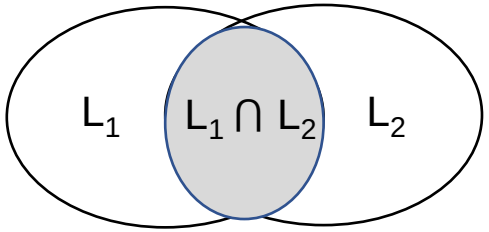
Formalmente, para todos los estados q y q´, símbolos s y s´, y movimientos d de {L, R, S} de la MT M_L:

- Si $\delta(q, s) = (q_A, s', d)$, entonces $\delta'(q, s) = (q_R, s', d)$
- Si $\delta(q, s) = (q_R, s', d)$, entonces $\delta'(q, s) = (q_A, s', d)$
- Si $\delta(q, s) = (q', s', d)$, con $q' \neq q_A$ y $q' \neq q_R$, entonces $\delta'(q, s) = (q', s', d)$

Claramente, $\mathbf{M_L{}^C}$ **acepta** $\mathbf{L^C}$ **y para siempre**, y por lo tanto $\mathbf{L^C \in R}$, que era lo que queríamos demostrar.

Lema 2. **Si $L_1 \in R$ y $L_2 \in R$, entonces $L_1 \cap L_2 \in R$**, tal que $L_1 \cap L_2$ es la intersección de L_1 y L_2.

Definición: $L_1 \cap L_2$ tiene las cadenas que están en L_1 y L_2.

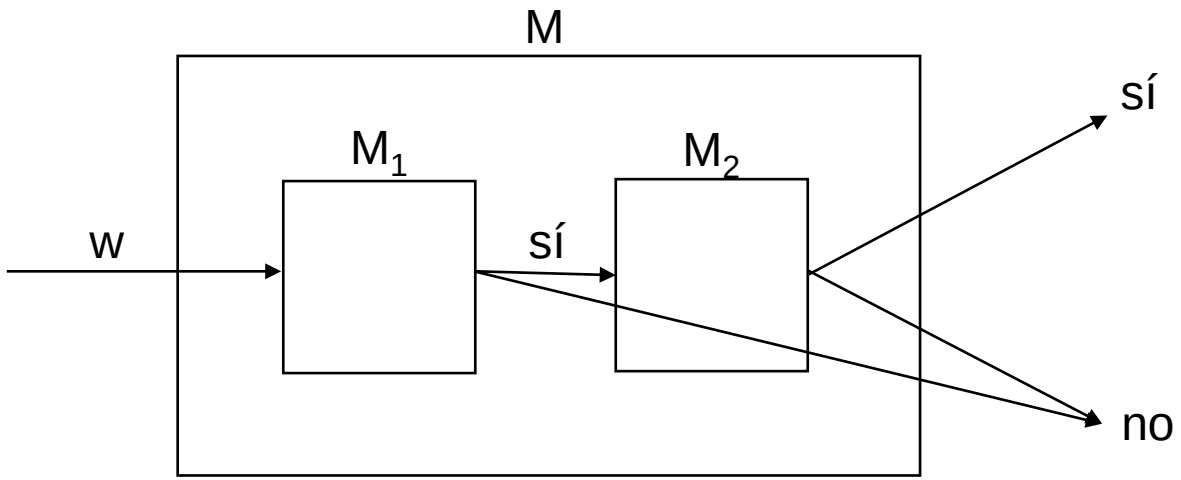


Prueba.

1. Idea general.

Dadas dos MT M_1 y M_2 que respectivamente aceptan L_1 y L_2 y paran siempre, la idea es construir una MT M que acepte $L_1 \cap L_2$ y pare siempre.

Propuesta de solución: ejecutar secuencialmente las dos MT, M_1 y M_2:



M acepta sólo las cadenas que pasan por los "filtros" de M_1 y M_2.

M para siempre porque M_1 y M_2 paran siempre.

Por lo tanto, $L_1 \cap L_2 \in R$.

2. Idea de construcción de la MT M.

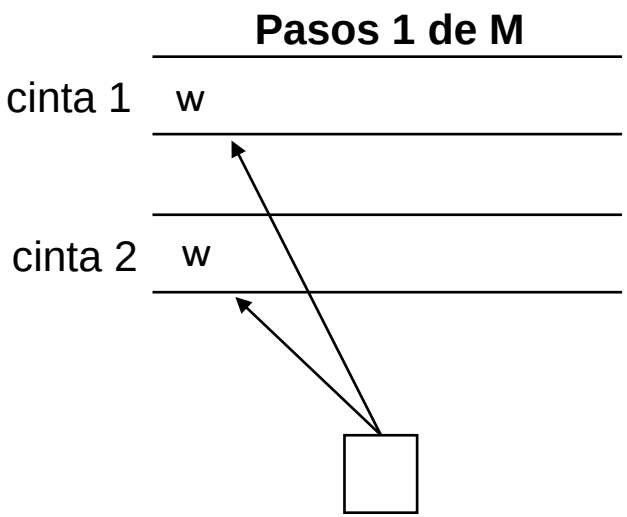
M tiene 2 cintas.
Dada la entrada w en la cinta 1, M hace:

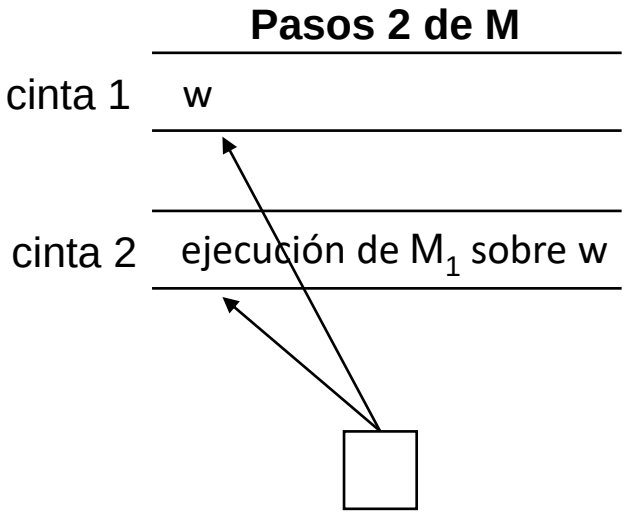
1. Copia w en la cinta 2.

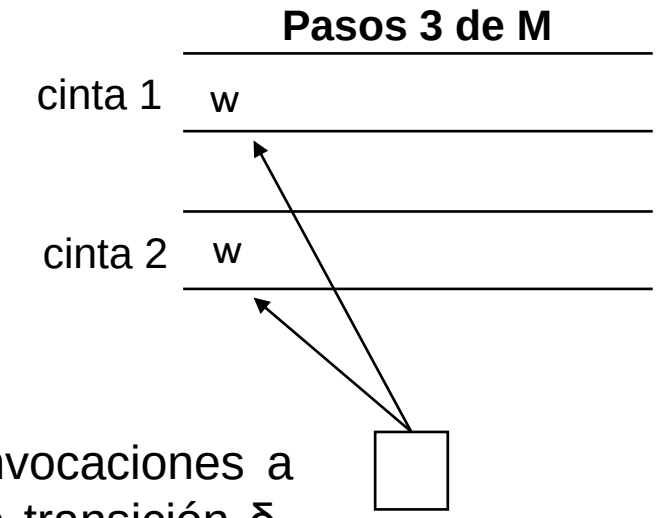
2. Ejecuta M_1 sobre w en la cinta 2.
   Si M_1 para en q_R,
   entonces para en q_R.

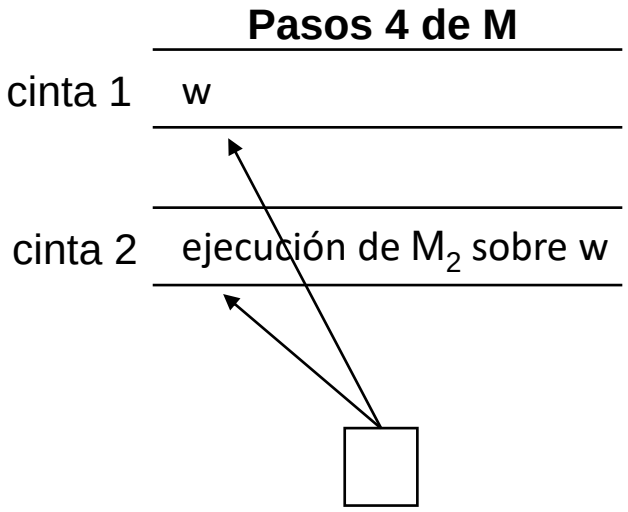
3. Borra el contenido de la cinta 2
   y copia de nuevo w en la cinta 2.

4. Ejecuta M_2 sobre w en la cinta 2.
   Si M_2 para en q_A (q_R),
   entonces para en q_A (q_R).









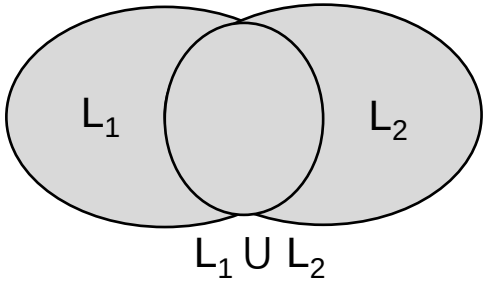
Los pasos 2 y 4 pueden entenderse como invocaciones a rutinas, que no son más que las funciones de transición δ_1 de M_1 y δ_2 de M_2.

También se cumple que si $L_1 \in R$ y $L_2 \in R$, entonces $L_1 \cup L_2 \in R$ (ejercicio).

Algunas propiedades de la clase RE

Lema 3. **Si $L_1 \in RE$ y $L_2 \in RE$, entonces $L_1 \cup L_2 \in RE$**, tal que $L_1 \cup L_2$ es la unión de L_1 y L_2.

Definición: $L_1 \cup L_2$ tiene las cadenas que están en L_1 o L_2.

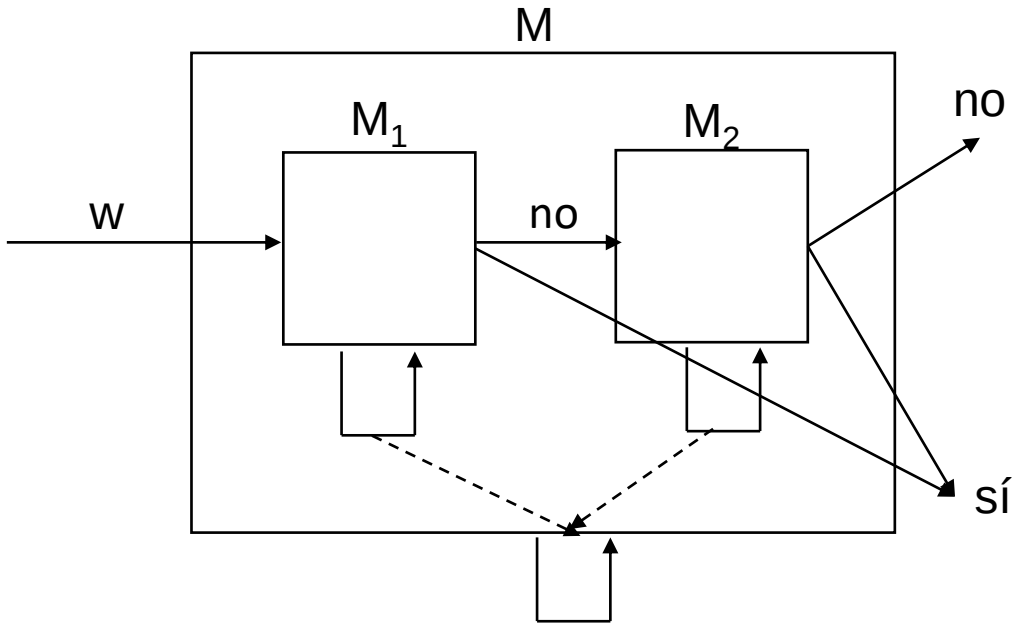


Prueba.

1. Idea general.

Dadas dos MT M_1 y M_2 que respectivamente aceptan L_1 y L_2, la idea es construir una MT M que acepte $L_1 \cup L_2$.

(Primera) propuesta de solución: como antes, ejecutar secuencialmente las dos MT:



¿Es correcta esta solución?

¡No! Si M_2 acepta w,

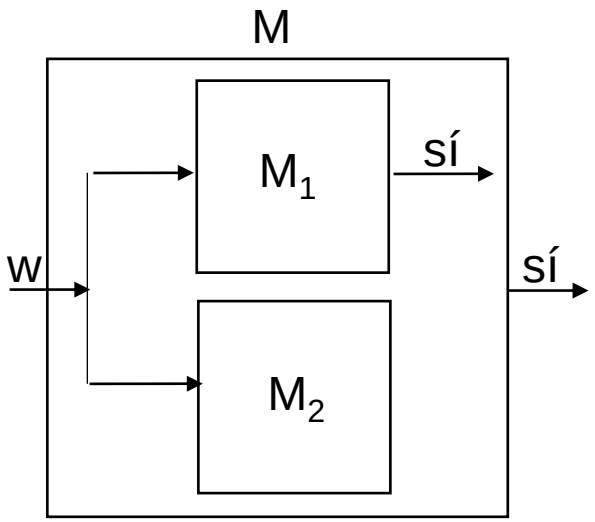
y M_1 no para sobre w,

entonces M, que debe aceptar w,

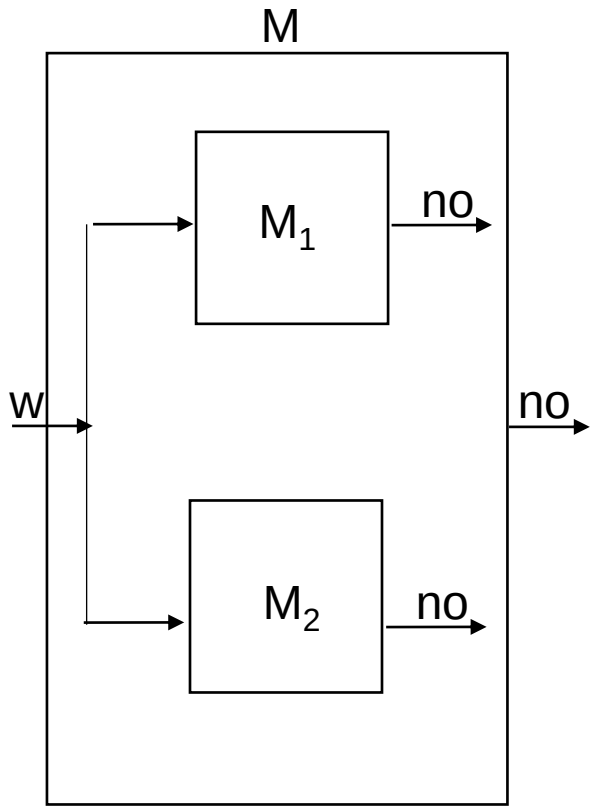
no lo hace.

- La forma correcta es ejecutar **“en paralelo”** M_1 y M_2. M acepta w sii una de las dos MT lo acepta.

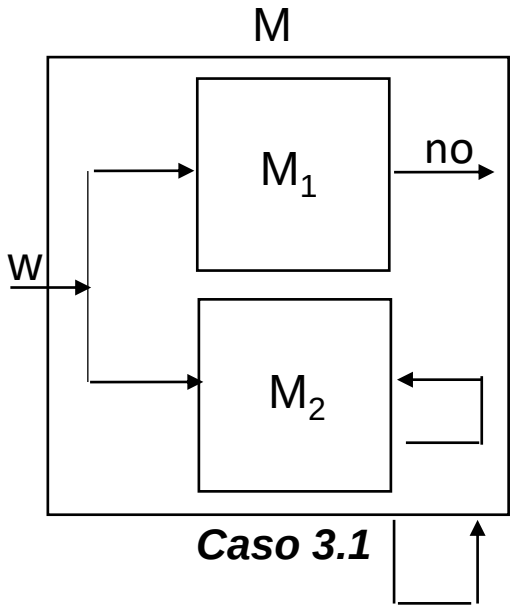
  (Ejecutar en paralelo M_1 y M_2 es simplemente ejecutar alternativamente un paso de cada una de las MT)



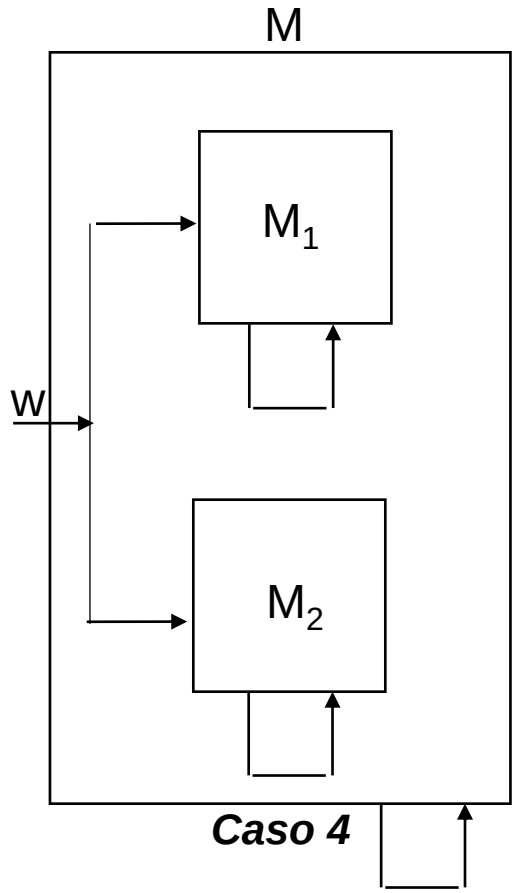
Caso 1.1



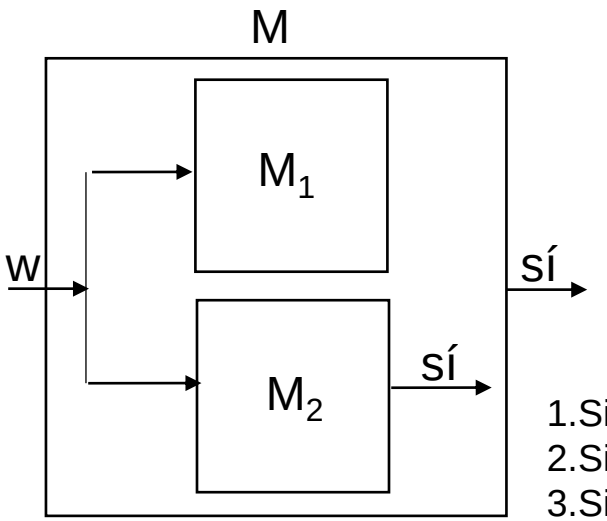
Caso 2



Caso 3.1

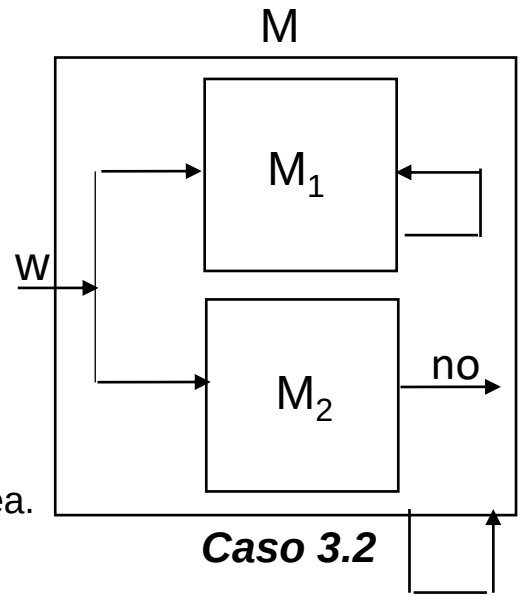


Caso 4



Caso 1.2

1. Si alguna de las dos MT acepta, M acepta.
2. Si las dos MT rechazan, M rechaza.
3. Si una MT rechaza y la otra loopea, M loopea.
4. Si las dos MT loopean, M loopea.



Caso 3.2

2. Idea de construcción de la MT M.

M tiene 3 cintas.
En la cinta 1 tiene la entrada w.
En las cintas 2 y 3 ejecuta M_1 y M_2, respectivamente.

La MT M hace:

1. Copia w de la cinta 1 a las cintas 2 y 3.

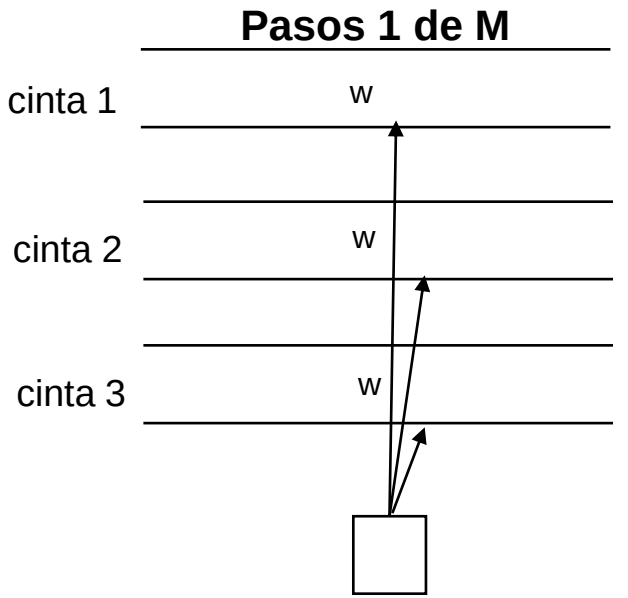
2. Ejecuta 1 paso de M_1 en la cinta 2. Si M_1 acepta, acepta.

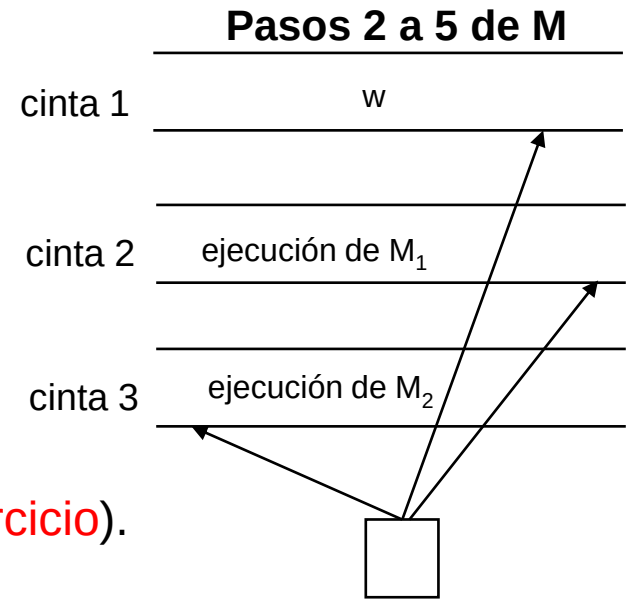
3. Ejecuta 1 paso de M_2 en la cinta 3. Si M_2 acepta, acepta.

4. Si M_1 y M_2 rechazan, rechaza.

5. Vuelve al paso 2.

- M acepta, rechaza o no para.
- En cada iteración memoriza los estados y posiciones de las 2 ejecuciones.



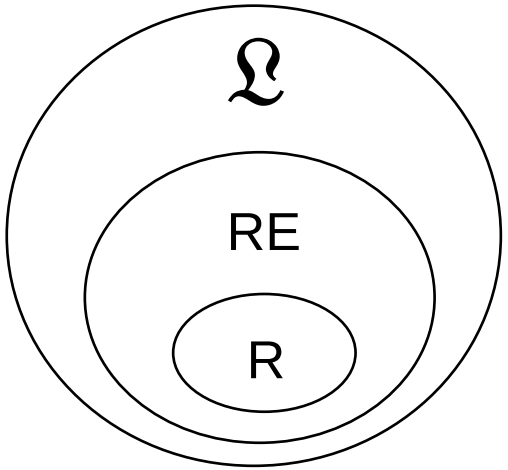


También se cumple que si $L_1 \in$ RE y $L_2 \in$ RE, entonces $L_1 \cap L_2 \in$ RE (ejercicio).

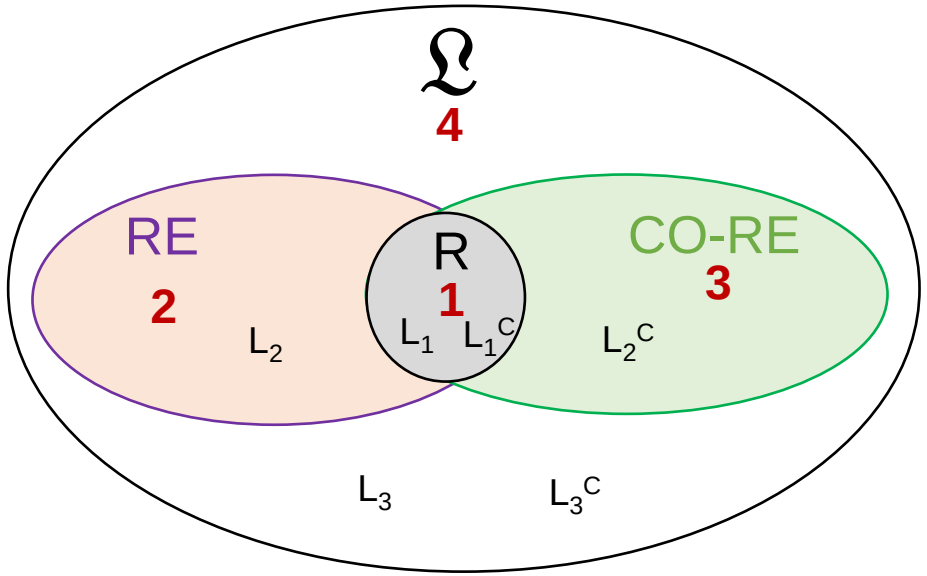
Por otro lado, $L \in$ RE NO IMPLICA que $L^c \in$ RE (diferencia con R).

Segunda versión de la jerarquía de la computabilidad

Primera versión



Segunda versión



CO-RE tiene los complementos de los lenguajes de RE

Región 1 (los lenguajes más "fáciles").
R es la clase de los lenguajes recursivos.
Si L_1 está en R, entonces también L_1^C está en R.

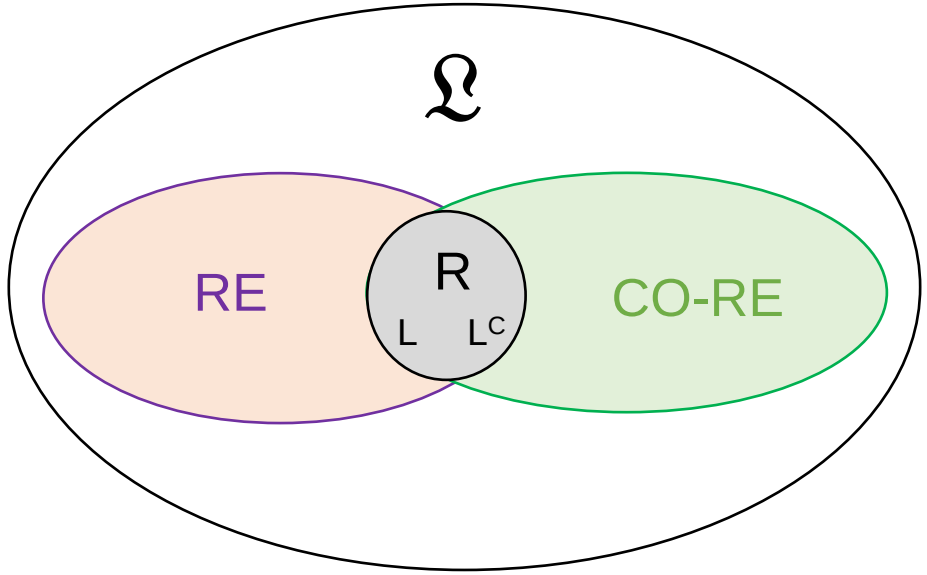
Región 2.
Clase de los lenguajes (RE – R).
Si L_2 está en RE, entonces L_2^C está en CO-RE.

Región 3.
Clase de los lenguajes (CO-RE – R).
Si L_2 está en CO-RE, entonces L_2^C está en RE.

Región 4 (los lenguajes más "difíciles").
Clase de los lenguajes Ω – (RE U CO-RE).
Si L_3 está en la clase, también está L_3^C.

Lema 4. **R = RE ∩ CO-RE.**

Es decir: L ∈ R sii (L ∈ RE y L ∈ CO-RE) sii (L ∈ RE y L^C ∈ RE).



Los lenguajes de CO-RE son los complementos de los lenguajes de RE

a) **R ⊆ RE ∩ CO-RE:** Si L ∈ R, entonces:
L ∈ RE por definición.
L^C ∈ R por el Lema 1, y entonces **L^C ∈ RE** por definición.
Por lo tanto, L ∈ RE y L^C ∈ RE, es decir L ∈ RE y L ∈ CO-RE,
es **decir L ∈ RE ∩ CO-RE.**

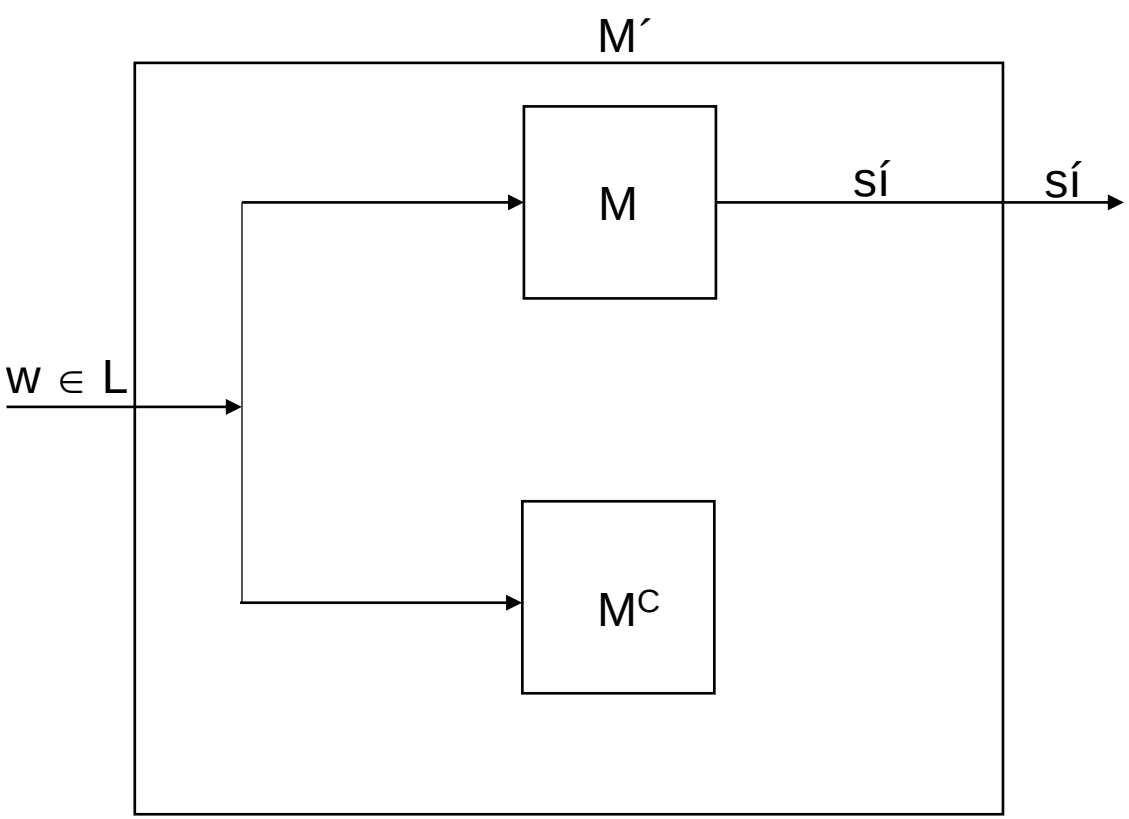
b) **RE ∩ CO-RE ⊆ R:** Si L ∈ RE ∩ CO-RE, entonces:
L ∈ RE y L ∈ CO-RE.
L ∈ RE y L^C ∈ RE.
Existe una MT M que acepta L y existe una MT M^C que acepta L^C.
Mostramos en la próxima hoja cómo construir a partir de M y M^C una MT M´ que acepta L y para siempre,
lo que prueba que **L ∈ R**.

CONCLUSIÓN: Contar con dos MT, una para aceptar L (puede no parar siempre) y otra para aceptar L^C (puede no parar siempre), permite contar con una MT para decidir L (lo acepta y para siempre).

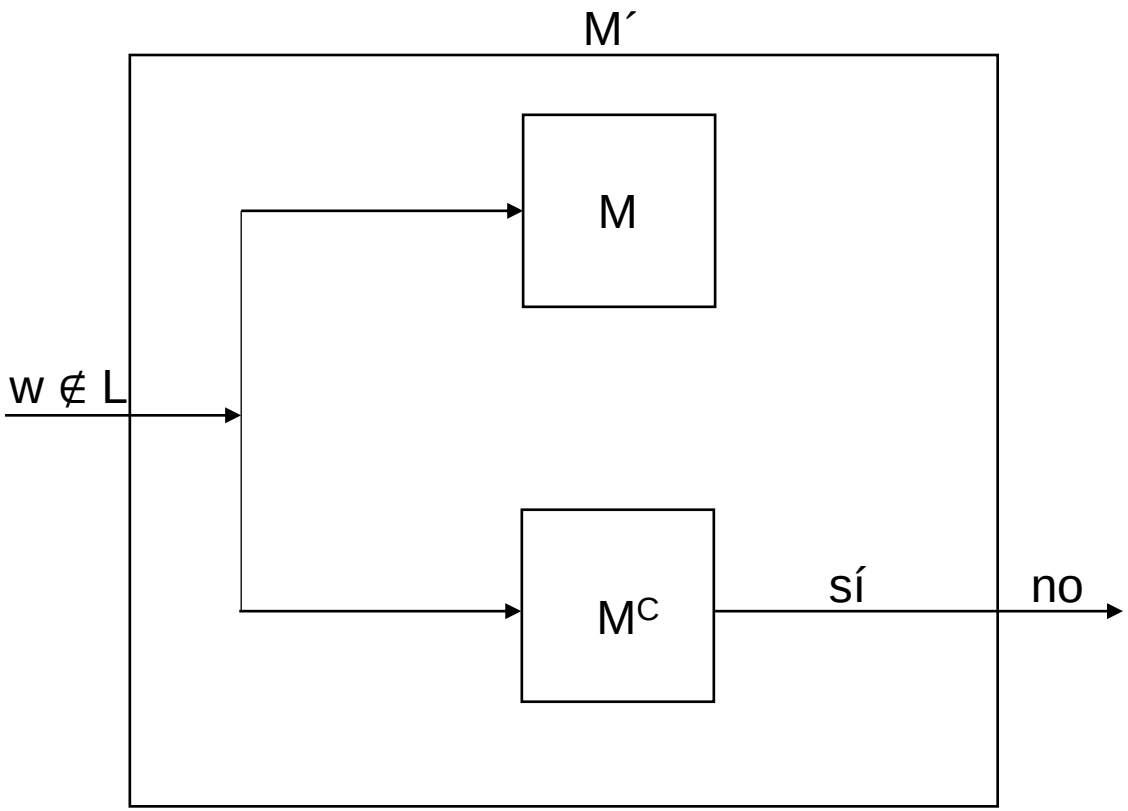
Idea general de la prueba de RE ∩ CO-RE ⊂ R.

Hipótesis: existen una MT M que acepta L y una MT M^C que acepta L^C.

Construimos una MT M´ que ejecuta "en paralelo" M y M^C:



Caso 1



Caso 2

Se cumple que M´ acepta L y para siempre:

- Para todo w, **w ∈ L** o **w ∈ L^C**. Así, **M acepta w** o **M^C acepta w,** respectivamente. Por lo tanto, M´ para siempre.
- M´ acepta L porque acepta una cadena w sii M acepta w.

Anexo de la clase teórica 2

Jerarquía de la computabilidad

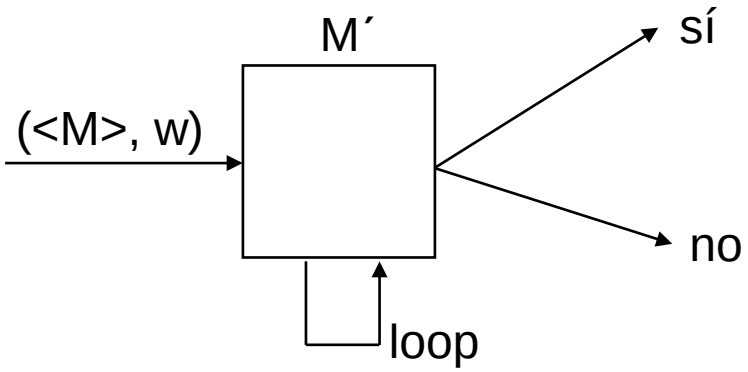
Ejemplos clásicos de lenguajes no recursivos

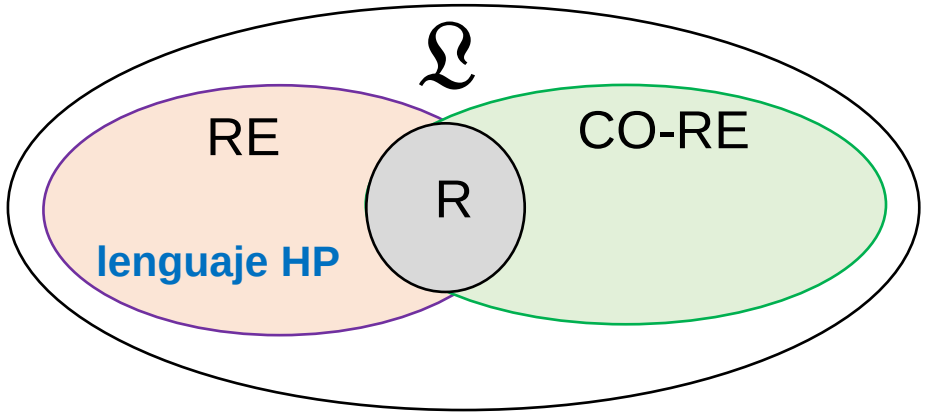
Problema de la parada de una MT (Halting Problem)

Dada una MT M y una cadena w, ¿M para a partir de w?

El lenguaje **HP = {(<M>, w) | M para a partir de w}** pertenece al conjunto **RE – R**.

A. Turing lo demostró en 1936.





<M> es el código de una MT.

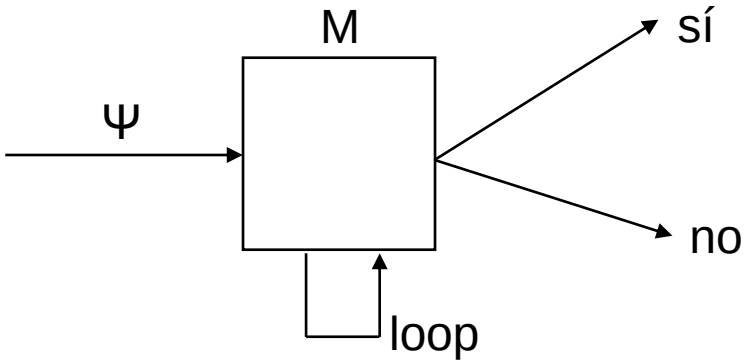
Ejercicio: ¿cómo sería una MT M´ que acepta el lenguaje HP?

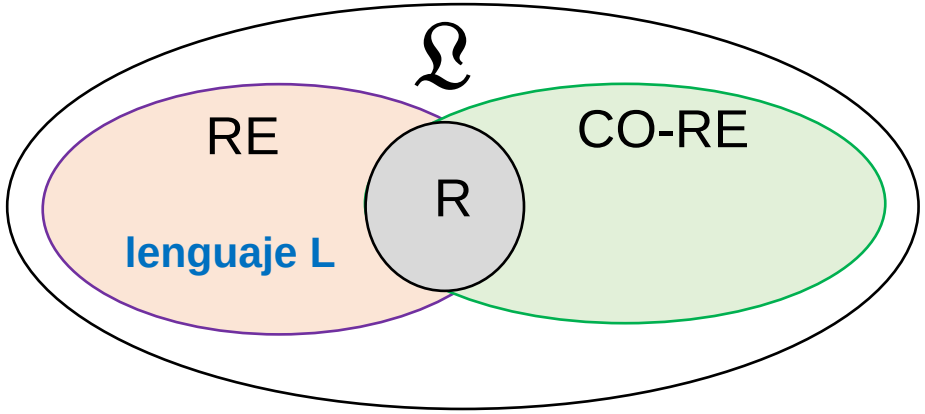
Problema de resolución de las ecuaciones diofánticas

Dada una ecuación algebraica con coeficientes enteros y variables enteras (conocida como ecuación diofántica), como por ejemplo $2x^3 + 5y^3 = 6z^3$, ¿tiene solución?

El lenguaje **L = {Ψ | Ψ es una ecuación diofántica y tiene solución}** pertenece al conjunto **RE – R**.

Fue el décimo de los famosos 23 problemas formulados por D. Hilbert en 1900 que desarrollaron fuertemente las matemáticas del siglo XX. En 1970, Y. Matiyasevich demostró que el lenguaje L no es recursivo.





Problema de decisión en la lógica de predicados (Entscheidungsproblem)

Dada una fórmula de la lógica de predicados Φ, ¿Φ es un teorema?

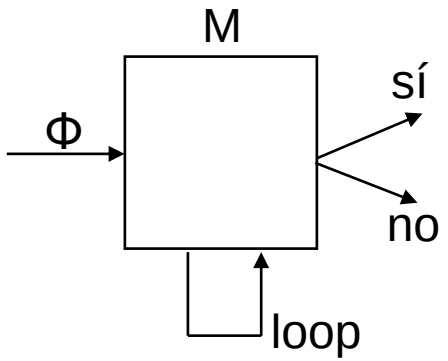
El lenguaje **L = {Φ | Φ es un teorema de la lógica de predicados}** pertenece al conjunto **RE – R**.

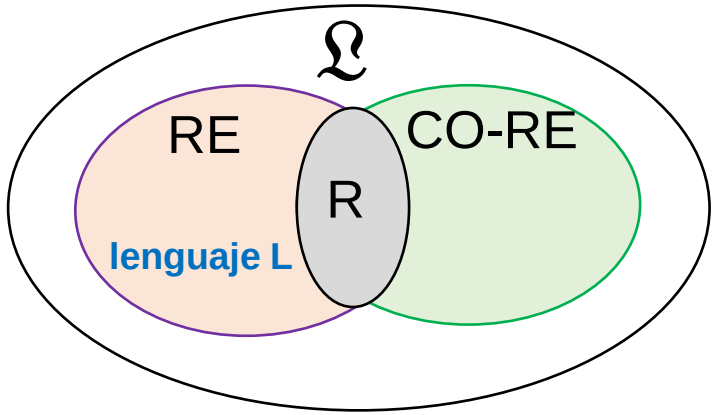
A. Turing lo demostró en 1936 (otro problema planteado por D. Hilbert, en este caso a fines de la década de 1920).

EJEMPLO

Axiomas y Reglas

K_1 : $A \rightarrow (B \rightarrow A)$
K_2 : $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
K_3 : $(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$ K_4 : $(\forall x)\, A(x) \rightarrow A(x|t)$
K_5 : $(\forall x)\, (A \rightarrow B) \rightarrow (A \rightarrow (\forall x)\, B)$
Modus Ponens (MP): A y $A \rightarrow B$ implican B
Generalización: A implica $(\forall x)$ A





Prueba del Teorema: Φ = $\neg P \rightarrow (P \rightarrow Q)$

1. $\neg P \rightarrow (\neg Q \rightarrow \neg P)$	axioma K_1
2. $(\neg Q \rightarrow \neg P) \rightarrow (P \rightarrow Q)$	axioma K_1
3. $((\neg Q \rightarrow \neg P) \rightarrow (P \rightarrow Q)) \rightarrow (\neg P \rightarrow ((\neg Q \rightarrow \neg P) \rightarrow (P \rightarrow Q)))$	axioma K_1
4. $\neg P \rightarrow ((\neg Q \rightarrow \neg P) \rightarrow (P \rightarrow Q))$	MP 2 y 3
5. $(\neg P \rightarrow ((\neg Q \rightarrow \neg P) \rightarrow (P \rightarrow Q))) \rightarrow ((\neg P \rightarrow (\neg Q \rightarrow \neg P)) \rightarrow (\neg P \rightarrow (P \rightarrow Q)))$	axioma K_2
6. $(\neg P \rightarrow (\neg Q \rightarrow \neg P)) \rightarrow (\neg P \rightarrow (P \rightarrow Q))$	MP 4 y 5
7. **$\neg P \rightarrow (P \rightarrow Q)$**	MP 1 y 6

Problema de decisión en la aritmética

Dada una fórmula de la aritmética Θ, ¿Θ es verdadera?

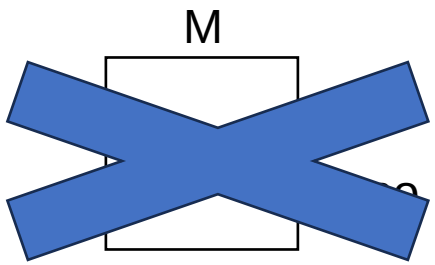
El lenguaje **L = {Θ | Θ es una fórmula verdadera de la aritmética}** pertenece a **Ω – (RE ∪ CO-RE).**

K. Gödel demostró este teorema, conocido como Teorema de Incompletitud, en 1931.

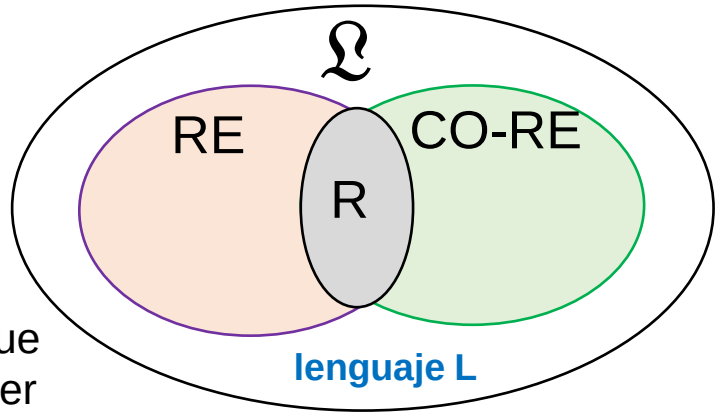
EJEMPLO

Axiomas y Reglas

K_1 : $A \rightarrow (B \rightarrow A)$
K_2 : $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
K_3 : $(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$
K_4 : $(\forall x)\, A(x) \rightarrow A(x|t)$
K_5 : $(\forall x)\, (A \rightarrow B) \rightarrow (A \rightarrow (\forall x)\, B)$
K_6 a K_{10}: Axiomas de la Igualdad
N_1 : $(\forall x) \neg(s(x) = 0)$
N_2 : $(\forall x)(\forall y)(x = y \rightarrow s(x) = s(y))$
N_3 : $(\forall x)(x + 0 = x)$
N_4 : $(\forall x)(\forall y)(x + s(y) = s(x + y))$
N_5 : $(\forall x)\, (x \cdot 0 = 0)$
N_6 : $(\forall x)(\forall y)(x \cdot s(y) = x \cdot y + x)$
N_7 : $P(0) \rightarrow ((\forall x)(P(x) \rightarrow P(s(x))) \rightarrow (\forall x)\, P(x))$
Modus Ponens (MP): A y $A \rightarrow B$ implican B
Generalización: A implica $(\forall x)$ A



Hay fórmulas verdaderas que ninguna MT puede reconocer



Prueba de la fórmula: Θ = (1 + 1 = 2) (extracto)

1. $(\forall x)(x + 0 = x)$	axioma N_3
2. $(\forall x)(x + 0 = x) \rightarrow 1 + 0 = 1$	axioma K_4
3. $1 + 0 = 1$	MP entre 1 y 2
.....	
14. $s(1 + 0) = s(1) \rightarrow 1 + s(0) = s(1)$	MP entre 8 y 13
15. $1 + s(0) = s(1)$	MP entre 11 y 14
16. $1 + 1 = 2$	

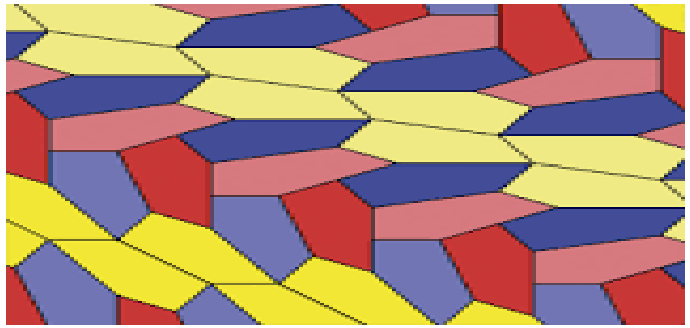
Problema de cubrimiento (o teselación) del plano

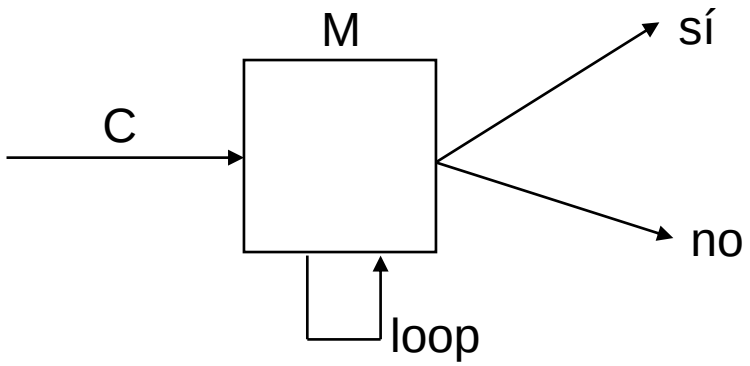
Dado un conjunto finito de figuras poligonales (o mosaicos), ¿con dichas figuras se puede cubrir el plano?

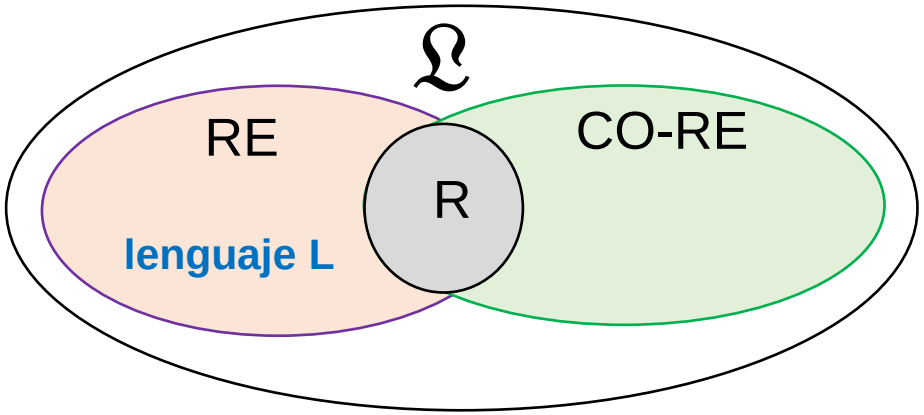
El lenguaje **L = {C | C es un conjunto finito de figuras poligonales que cubren el plano}** pertenece al conjunto **RE – R**.

R. Berger lo demostró en 1966.

EJEMPLO







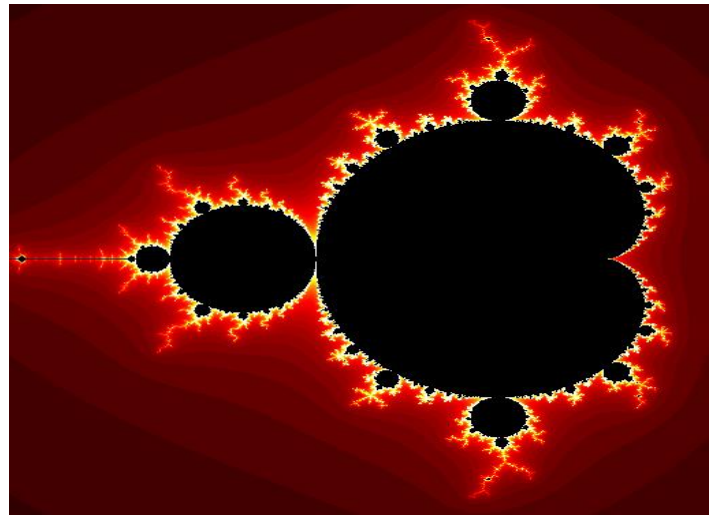
Problema de pertenencia al Conjunto de Mandelbrot

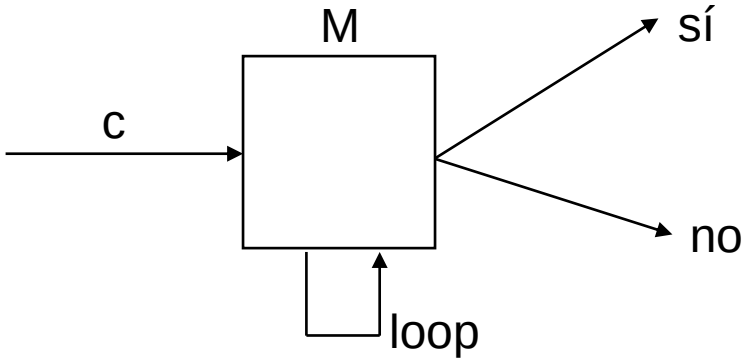
Dado un conjunto de puntos complejos en el plano definido por la sucesión $z_0 = 0$, $z_{n+1} = z_n^2 + c$, conjunto conocido como Conjunto de Mandelbrot (es un fractal), y dado un complejo c, ¿c pertenece al Conjunto de Mandelbrot?

El lenguaje **L = {c | c es un número complejo que está en el Conjunto de Mandelbrot}** pertenece a **CO-RE – R**.

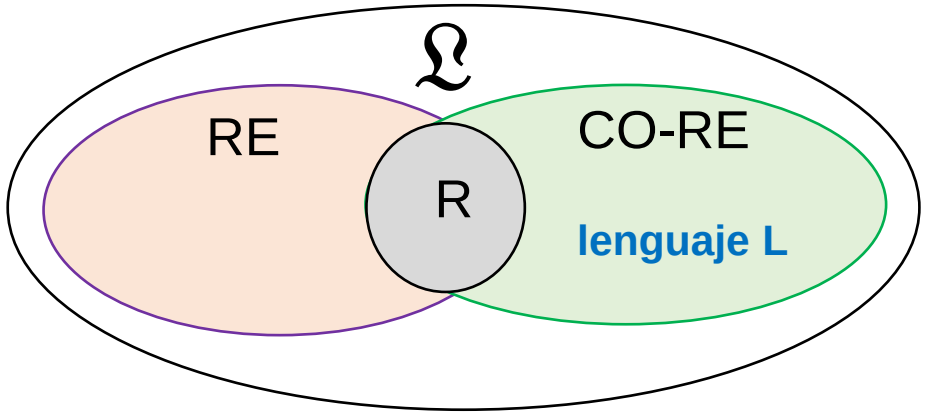
L. Blum lo demostró en 1989.

CONJUNTO DE MANDELBROT





Existe una MT M que acepta a todos los números complejos c que NO pertenecen al Conjunto de Mandelbrot



Clase práctica 2

Jerarquía de la computabilidad

Ejercicio 1. Probar que la clase R es cerrada con respecto a la operación de concatenación.
Es decir que si $L_1 \in R$ y $L_2 \in R$, entonces también $L_1 . L_2 \in R$.

Idea general.

El lenguaje $L_1 . L_2$ contiene todas las cadenas $w = x_1x_2$, tales que la subcadena $x_1 \in L_1$ y la subcadena $x_2 \in L_2$.

Sea M_1 una MT que decide el lenguaje L_1 y M_2 una MT que decide el lenguaje L_2.
Hay que construir una MT M que decida el lenguaje $L_1 . L_2$.

Dado un input w con n símbolos, M hace:

1. M ejecuta M_1 a partir de los primeros 0 símbolos de w, y M_2 a partir de los últimos n símbolos de w.
   Si en ambos casos se acepta, entonces M acepta.
2. Si no, M hace lo mismo que en (1) pero ahora con el 1er símbolo y los últimos (n – 1) símbolos de w.
   Si en ambos casos se acepta, entonces M acepta.
3. Si no, M hace lo mismo que en (1) pero ahora con los primeros 2 y los últimos (n – 2) símbolos de w.
   Si en ambos casos se acepta, entonces M acepta.

Y así siguiendo, con 3 y (n – 3), 4 y (n – 4), …, hasta llegar a n y 0 símbolos de w.
Si en ninguno de los casos se acepta, entonces M rechaza.

Queda como ejercicio la construcción de M y la verificación de su correctitud.

Ejercicio 2. Probar que también la clase RE es cerrada con respecto a la operación de concatenación, es decir que si $L_1 \in RE$ y $L_2 \in RE$, entonces también $L_1 . L_2 \in RE$.

Idea general.

Tal como se hizo con los lenguajes recursivos, se tiene que construir una MT M que reconozca $L_1 . L_2$ ejecutando sobre un input w (de n símbolos) determinadas MT M_1 y M_2 (MT que reconocen L_1 y L_2, respectivamente, las cuales ahora pueden loopear en casos negativos),
primero a partir de 0 y n símbolos de w,
después a partir de 1 y n – 1 símbolos de w,
y así siguiendo hasta llegar a n y 0 símbolos de w, aceptando eventualmente.

La diferencia con el caso de los lenguajes recursivos está en que ahora, teniendo en cuenta los posibles loops de M_1 y M_2, M debe ejecutarlas “en paralelo”:

M primero debe hacer ejecuciones de 1 paso de M_1 y M_2 con todas las posibles particiones de w,
luego ejecuciones de 2 pasos con todas las particiones,
luego ejecuciones de 3 pasos con todas las particiones,
y así siguiendo hasta eventualmente aceptar.

Queda como ejercicio la construcción de M y la verificación de su correctitud.

Ejercicio 3. Probar que la clase RE es cerrada con respecto a la operación de unión, permitiendo como solución una MT no determinística (MTN).

Idea general y construcción.

Sean dos lenguajes L_1 y L_2 de RE, aceptados por MT M_1 y M_2,
con $M_1 = (Q_1, \Sigma_1, \delta_1, q'_0, q_A, q_R)$ y $M_2 = (Q_2, \Sigma_2, \delta_2, q''_0, q_A, q_R)$.

Vamos a construir una MTN M que acepta $L_1 \cup L_2$:

Sea q_0 un estado que no está en Q_1 ni en Q_2. La MTN M es:

$M = (Q = Q_1 \cup Q_2 \cup \{q_0\}, \Sigma = \Sigma_1 \cup \Sigma_2, \Delta, q_0, q_A, q_R)$, tal que:

$\Delta = \delta_1 \cup \delta_2 \cup \{(q_0, s, q'_0, s, S), (q_0, s, q''_0, s, S)\}$, considerando todos los símbolos s de Σ.

Es decir, al comienzo la MTN M pasa no determinísticamente a la configuración inicial de M_1 o de M_2,
y después se comporta determinística como ellas.

Queda como ejercicio la verificación de la correctitud de la construcción de la MTN M.