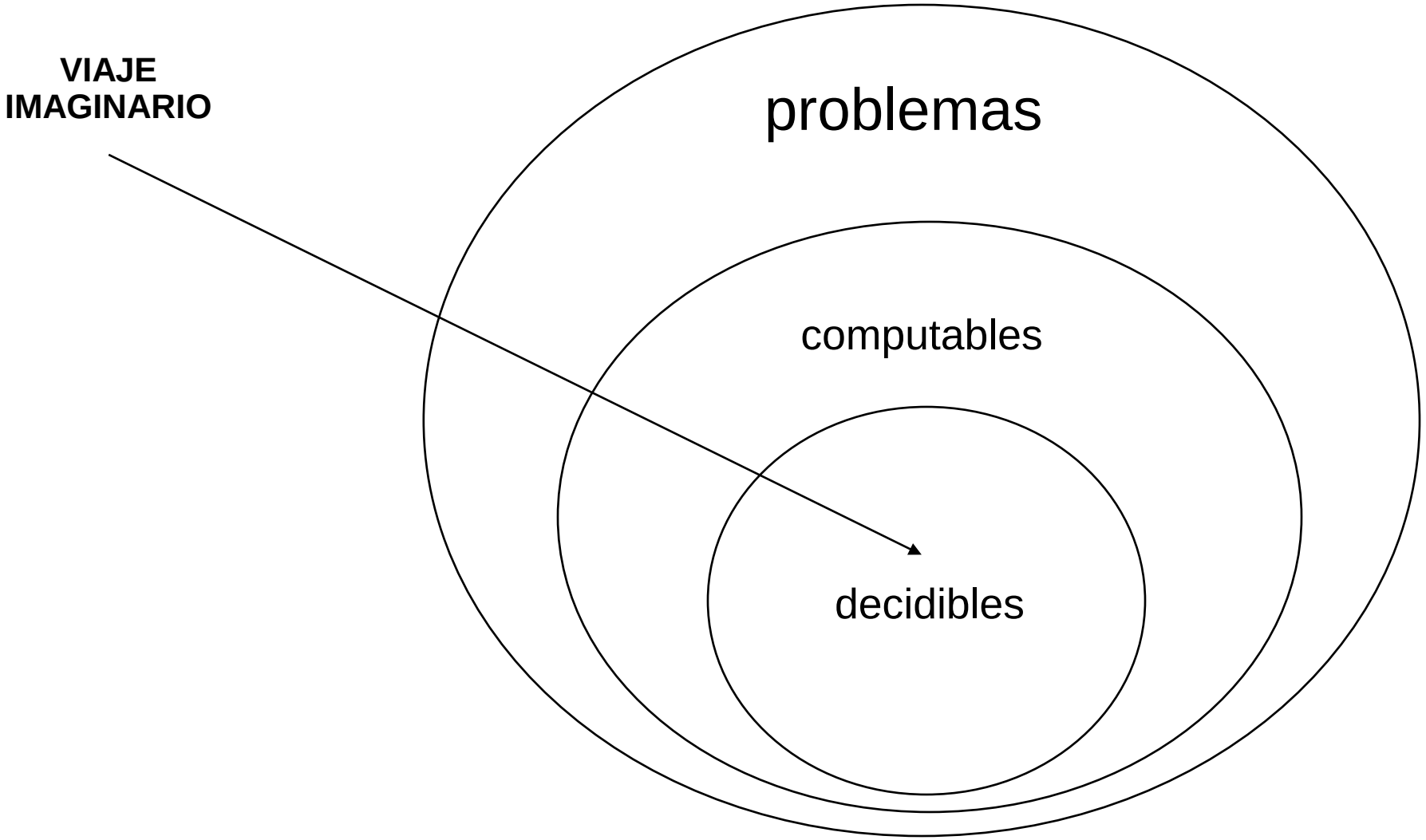
Clase teórica 2

La jerarquía de la computabilidad

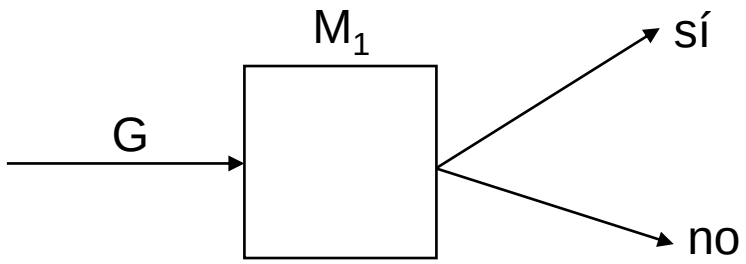
Iniciamos el viaje imaginario



Repaso

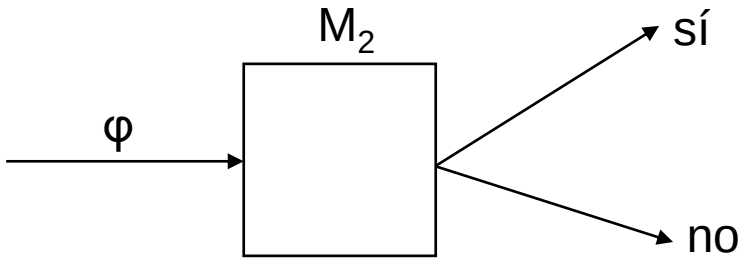
- Modelo computacional: **máquina de Turing (MT).**
- Problemas que consideraremos en general: **problemas de decisión o de tipo sí/no**.
- **problema "=" lenguaje** (cadenas que representan las instancias positivas del problema).
- MT que **acepta o reconoce un lenguaje**.

Ejemplos que vimos



Problema de si un grafo tiene un camino del primero al último vértice.

Una MT M_1 que resuelve el problema acepta el siguiente lenguaje:
$L(M_1)$ = {G | G es un grafo que tiene un camino del primero al último vértice}.

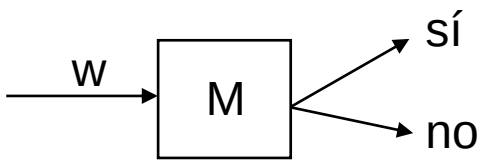


Problema de si una fórmula booleana es satisfactible.

Una MT M_2 que resuelve el problema acepta el siguiente lenguaje:
$L(M_2)$ = {φ | φ es una fórmula booleana satisfactible}.

Repaso y nuevas definiciones

- **Posibilidad 1.** Lenguajes aceptados por MT que **siempre paran**.



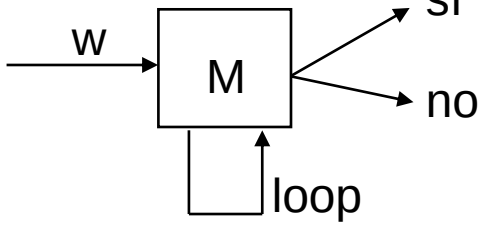
- Si $w \in L(M)$, M acepta.
- Si $w \notin L(M)$, M rechaza.

Problema computable decidible.

L es **recursivo.**

R es el conjunto de estos lenguajes.

- **Posibilidad 2.** Lenguajes aceptados por M que **a partir de algunas instancias negativas no paran**.



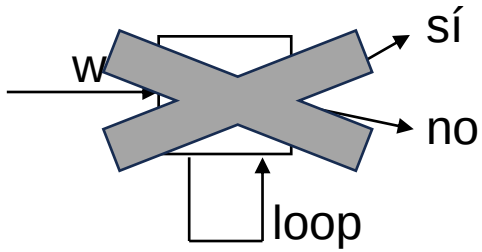
- Si $w \in L(M)$, M acepta.
- Si $w \notin L(M)$, M rechaza o loopea.

Problema computable no decidible.

L es **recursivamente enumerable.**

RE es el conjunto de estos lenguajes.

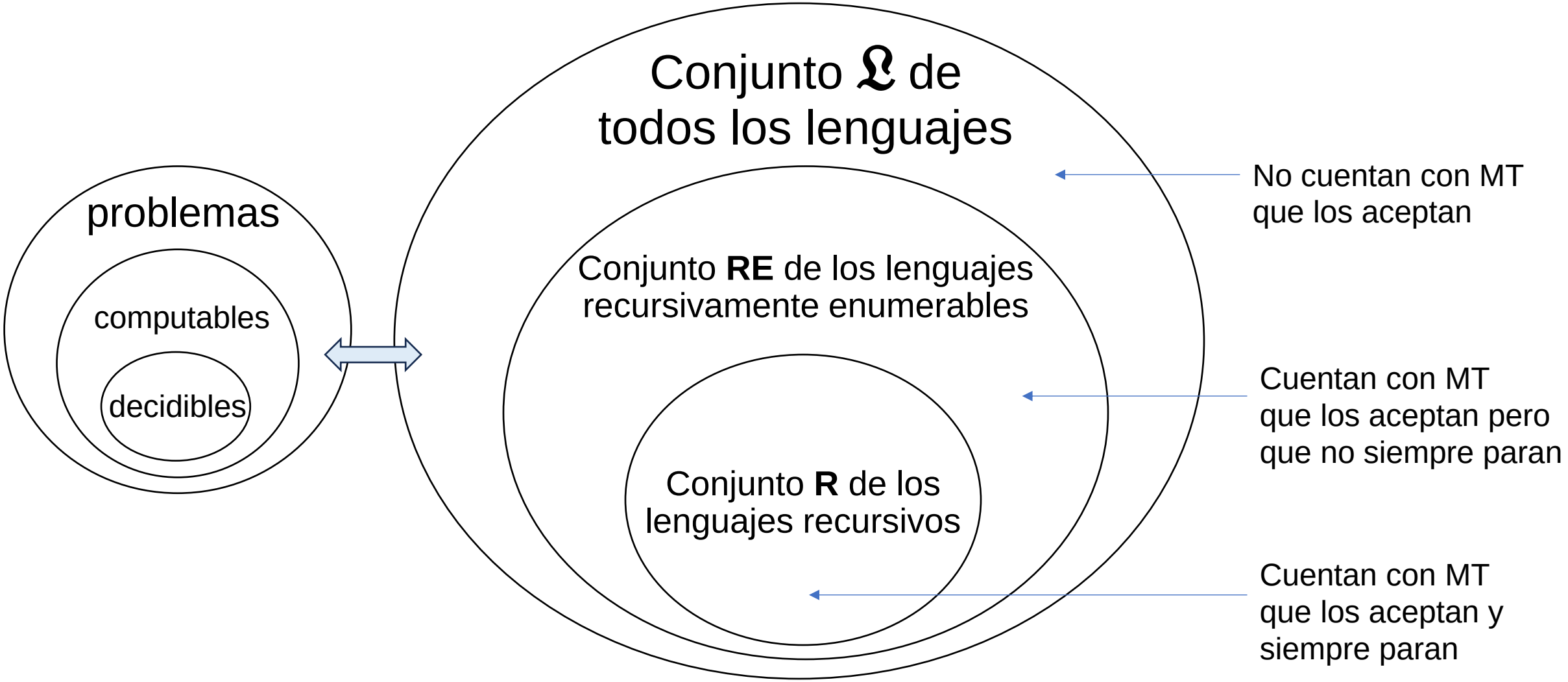
- **Posibilidad 3**. Lenguajes sin MT que los acepten (**a partir de algunas instancias positivas no paran).**



Problema no computable.

L no es **recursivamente enumerable**.

Primera versión de la jerarquía de la computabilidad



Formalizando las definiciones

- **Σ** es el alfabeto universal de todos los símbolos: Σ = {a, b, …, 1, 2, …, +, –, …}
- **Σ*** es el conjunto universal de todas las cadenas de símbolos de Σ: Σ* = {λ, a, b, 1, …, aa, ab, a1, …, aaa, …}
- $\mathfrak{L}$ es el conjunto universal de todos los lenguajes con alfabeto Σ: conjunto de todos los subconjuntos de Σ*.
- Un lenguaje L es **recursivo** (L ∈ R) sii existe una MT M que lo **acepta y siempre para (lo decide).**

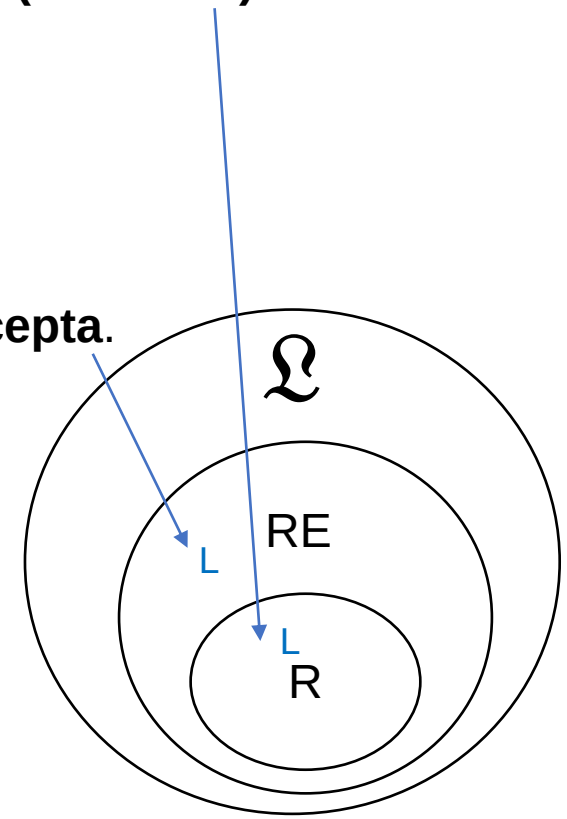
  Es decir, para toda cadena w de Σ*:

  - Si w ∈ L, entonces M a partir de w para en su estado q_A
  - Si w ∉ L, entonces M a partir de w para en su estado q_R

- Un lenguaje L es **recursivamente enumerable** (L ∈ RE) sii existe una MT M que lo **acepta**.

  Es decir, para toda cadena w de Σ*:

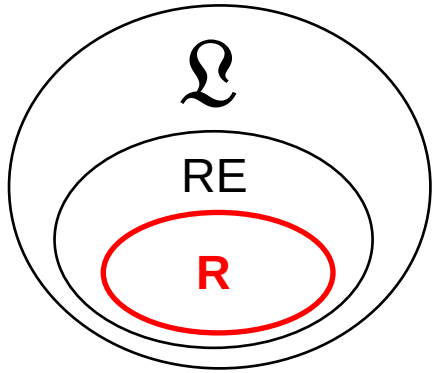
  - Si w ∈ L, entonces M a partir de w para en su estado q_A
  - Si w ∉ L, entonces M a partir de w para en su estado q_R o no para



Se cumple por definición que **R ⊆ RE ⊆** $\mathfrak{L}$ (ejercicio)

Las inclusiones son estrictas: R ⊂ RE ⊂ $\mathfrak{L}$ **(lo probamos en la próxima clase)**

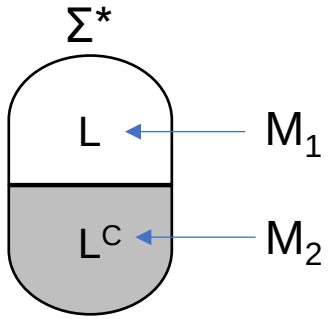
La clase R (lenguajes recursivos)



Propiedad 1. Si $L \in R$, entonces $L^C \in R$.

Es decir, si existe una MT M_1 que decide L,

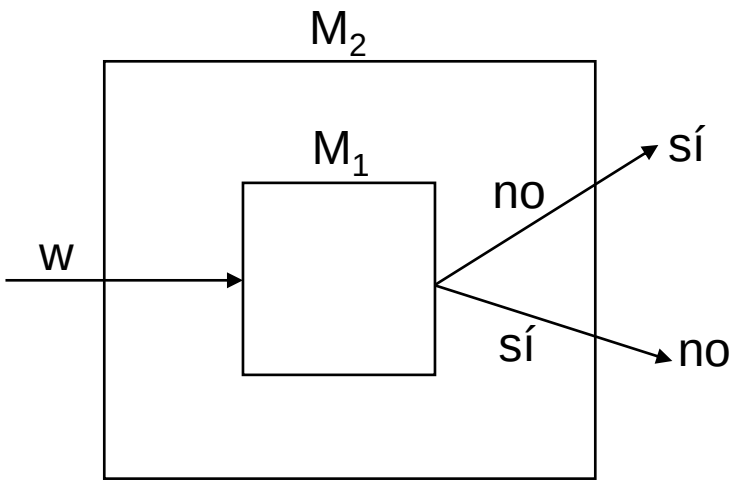
también existe una MT M_2 que decide L^C.



Prueba.

Idea general.

Construir una MT M_2 que responda al revés que la MT M_1.



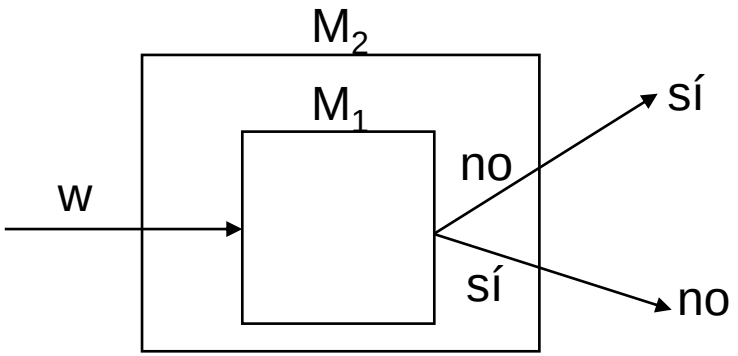
M_1 decide L (hipótesis).

M_2 decide L^C (construcción).

Construcción de la MT M_2.

Si: $M_1 = (Q, \Gamma, \delta, q_0, q_A, q_R)$

entonces: $M_2 = (Q, \Gamma, \delta', q_0, q_A, q_R)$



donde las funciones de transición δ y δ´ de M_1 y M_2 son iguales salvo que los estados q_A y q_R **están permutados.**

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

- Si $\delta(q, s) = (q_A, s', d)$, entonces $\mathbf{\delta'(q, s) = (q_R, s', d)}$ *** se cambia q_A por q_R
- Si $\delta(q, s) = (q_R, s', d)$, entonces $\mathbf{\delta'(q, s) = (q_A, s', d)}$ *** se cambia q_R por q_A
- Si $\delta(q, s) = (q', s', d)$, con $q' \neq q_A$ y $q' \neq q_R$, entonces $\mathbf{\delta'(q, s) = (q', s', d)}$ *** los otros casos quedan igual

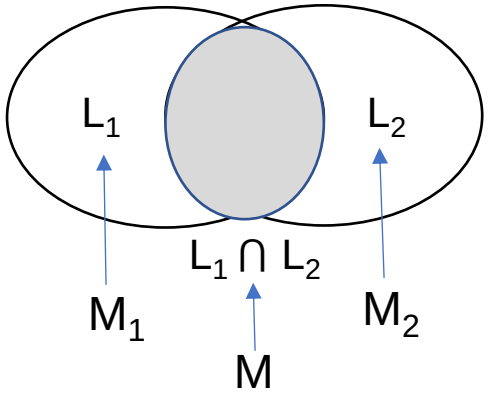
Conclusión: si un problema es decidible, entonces también lo es el problema contrario.

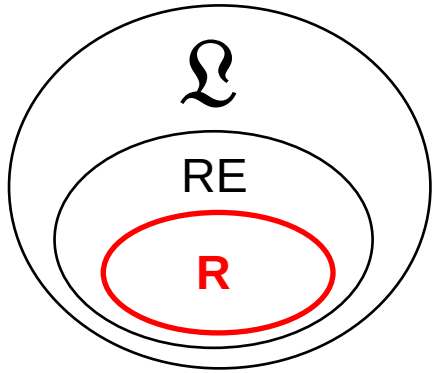
Propiedad 2. Si $L_1 \in R$ y $L_2 \in R$, entonces $L_1 \cap L_2 \in R$.

Es decir, si existe una MT M_1 que decide L_1,

y existe una MT M_2 que decide L_2,

también existe una MT M que decide $L_1 \cap L_2$.

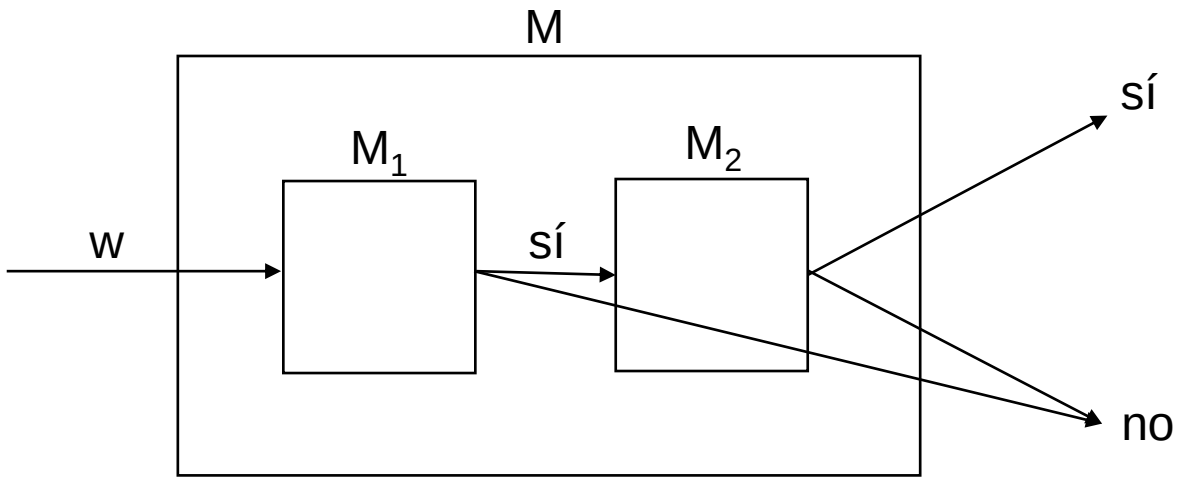




Prueba.

Idea general.

Construir una MT M que ejecute secuencialmente las MT M_1 y M_2 y acepte sii M_1 y M_2 aceptan.



M_1 decide L_1 (hipótesis).

M_2 decide L_2 (hipótesis).

M decide $L_1 \cap L_2$ (construcción).

Idea de la construcción de la MT M.

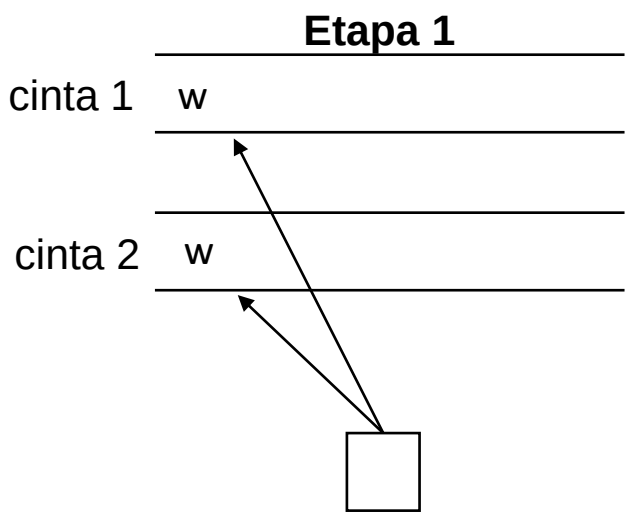
M tiene 2 cintas. Con w en la cinta 1, hace:

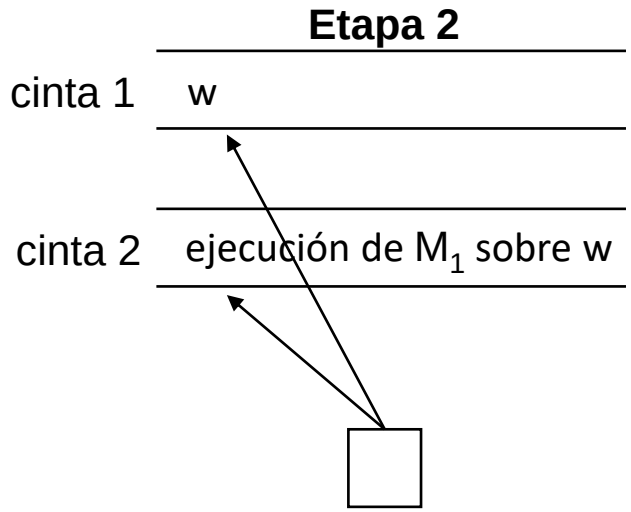
1. Copia w en la cinta 2.

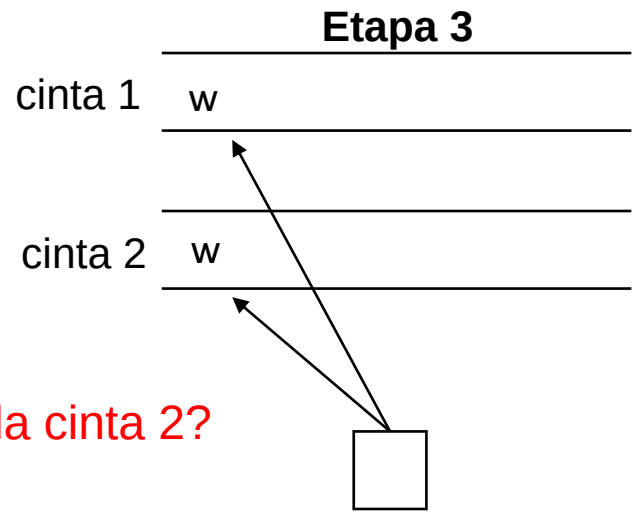
2. Ejecuta M_1 sobre w en la cinta 2.
   Si M_1 para en q_R,
   entonces M para en q_R.

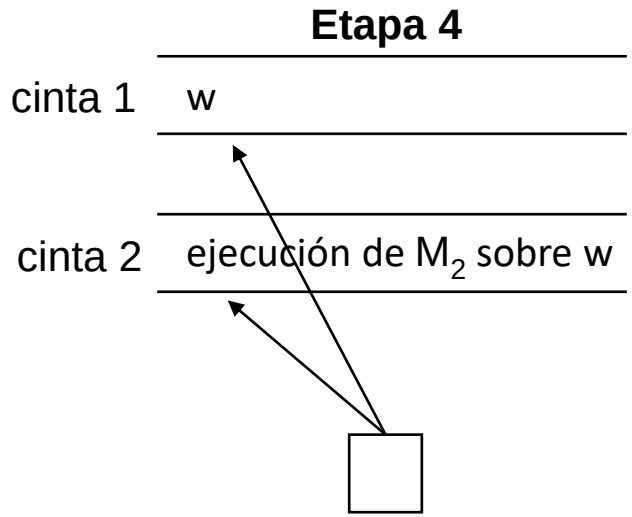
3. Borra el contenido de la cinta 2
   y copia de nuevo w en la cinta 2.

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







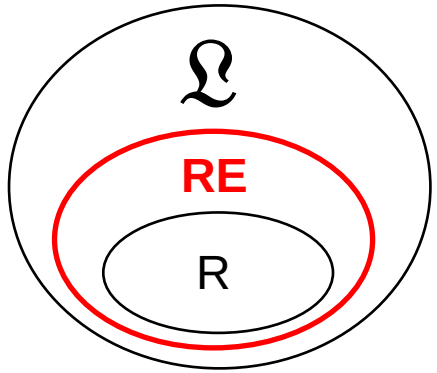


Ejercicio: ¿cómo hace M para copiar w y borrar la cinta 2?

Las etapas 2 y 4 pueden verse como invocaciones a rutinas, que no son más que las funciones de transición de M_1 y M_2. Es decir, el código de M puede verse de la siguiente manera: δ_M = …… δ_{M1} …… δ_{M2} ……

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

La clase RE (lenguajes recursivamente enumerables)

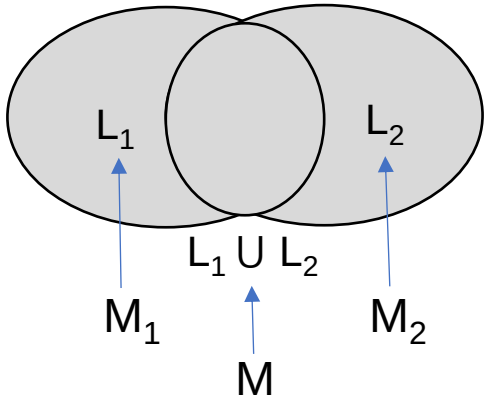


Propiedad 3. Si $L_1 \in RE$ y $L_2 \in RE$, entonces $L_1 \cup L_2 \in RE$.

Es decir, si existe una MT M_1 que acepta L_1,

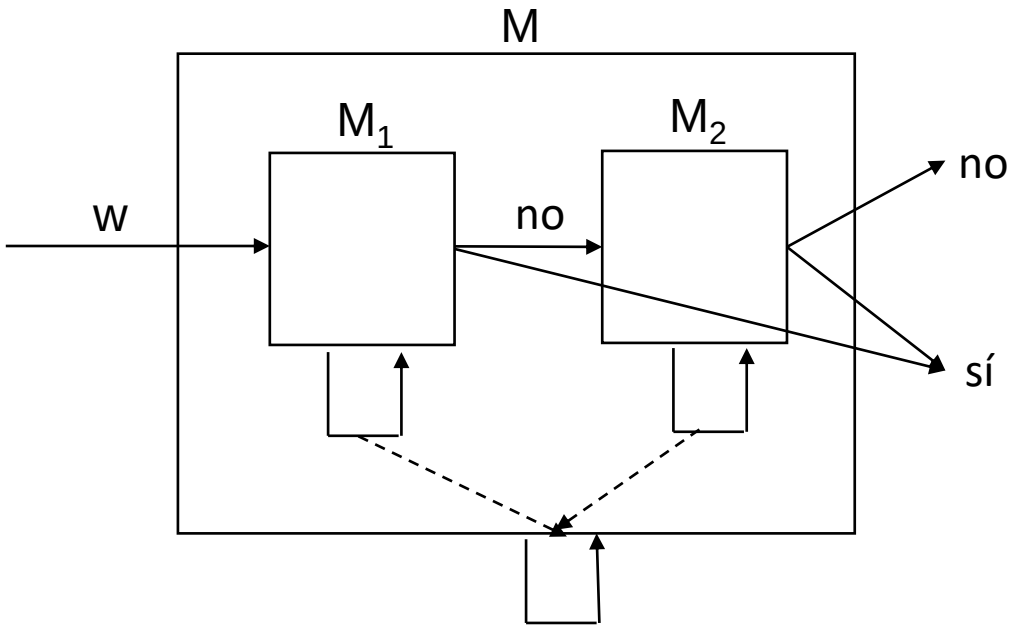
y existe una MT M_2 que acepta L_2,

también existe una MT M que acepta $L_1 \cup L_2$.



Prueba.

Idea general. ¿Construir una MT M que ejecute secuencialmente las MT M_1 y M_2 y acepte sii M_1 o M_2 aceptan?



¿Es correcta esta solución?

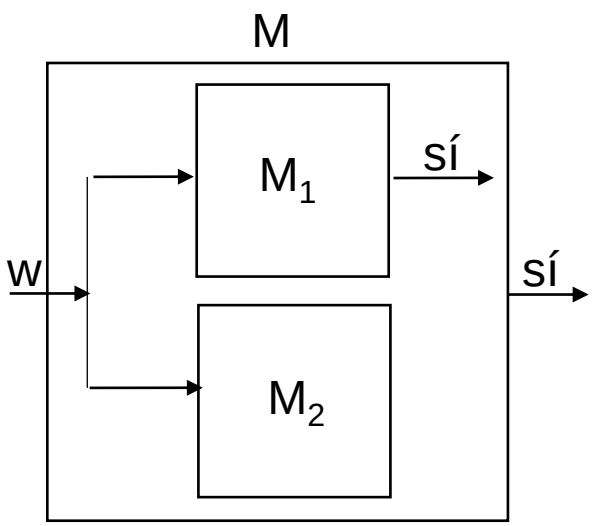
¡No!

Si M_2 acepta w, y M_1 no para a partir de w,

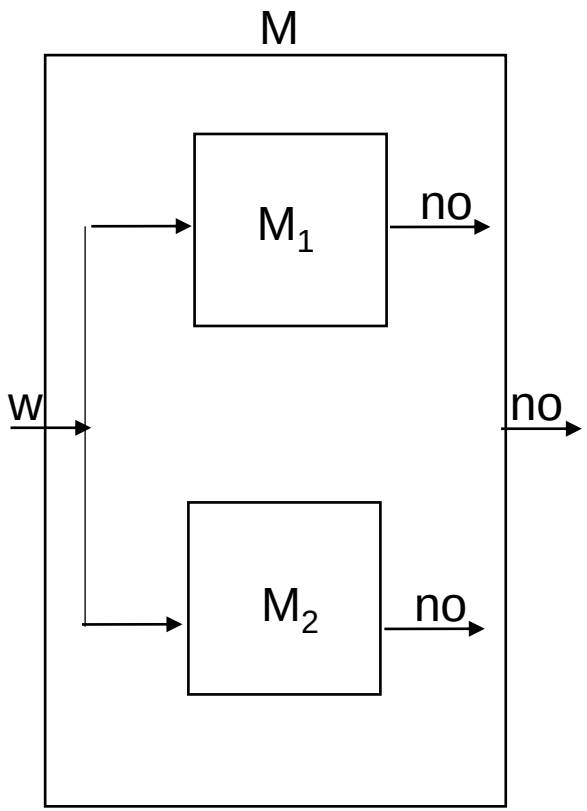
entonces M, que debe aceptar w, no lo hace.

- La forma correcta es ejecutar **"en paralelo"** M_1 y M_2, y aceptar w sii una de las dos MT acepta w.

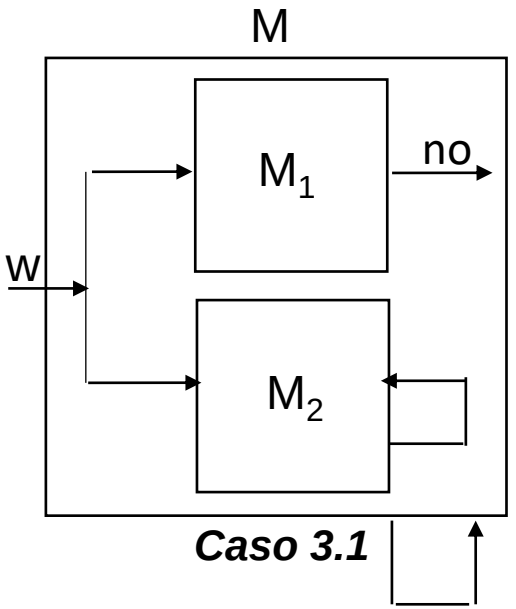
  "En paralelo" M_1 y M_2 = ejecutar un paso de M_1 y un paso de M_2 alternadamente.



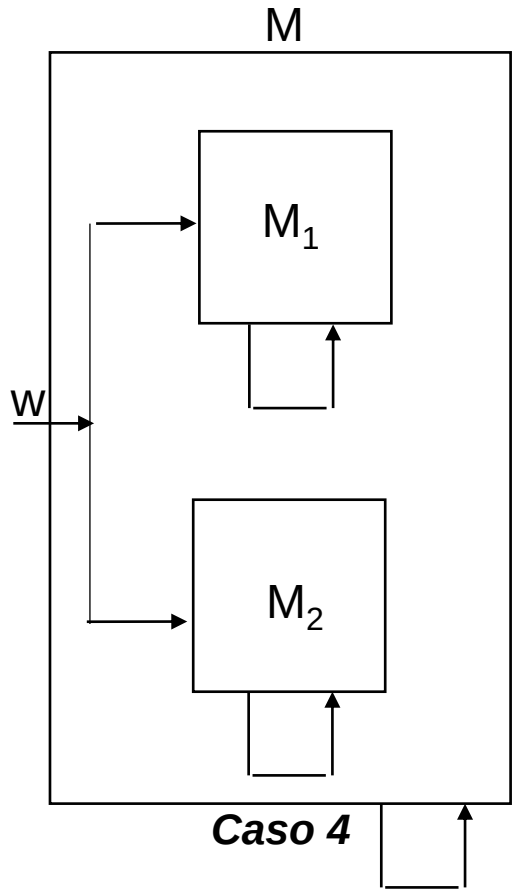
Caso 1.1



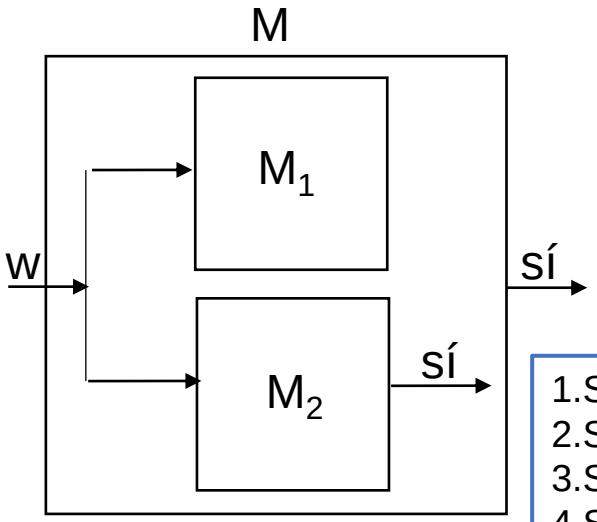
Caso 2



Caso 3.1

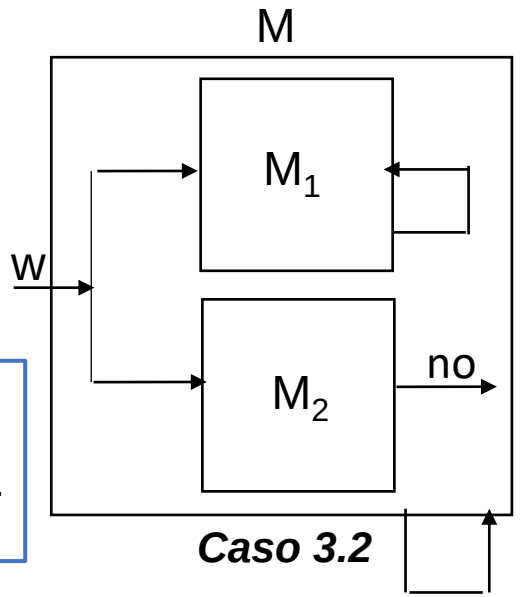


Caso 4



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

Idea de la 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:

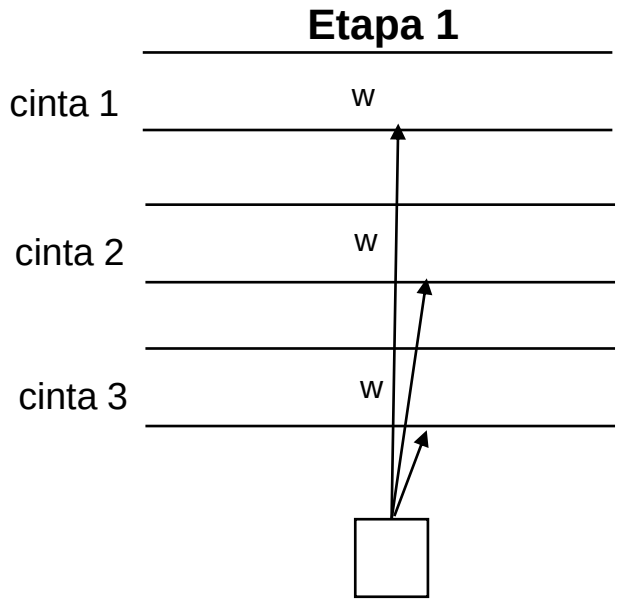
1. Copia w de la cinta 1 a las cintas 2 y 3.

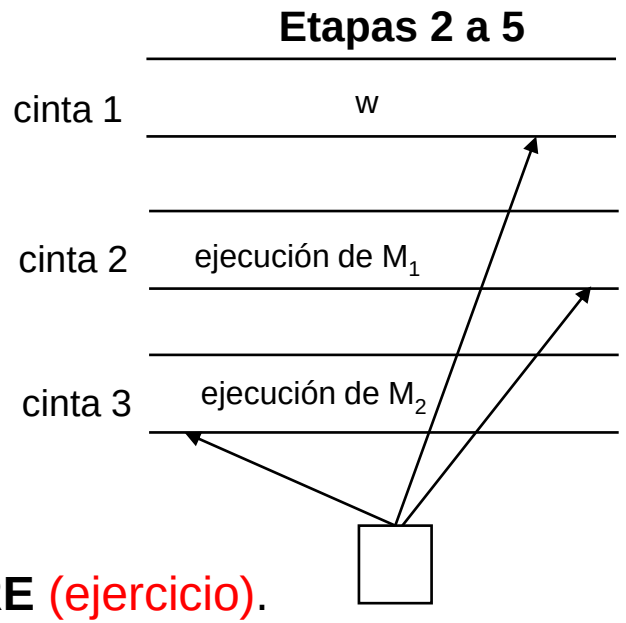
2. Ejecuta 1 paso de M_1 en la cinta 2. Si M_1 acepta, acepta.

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.



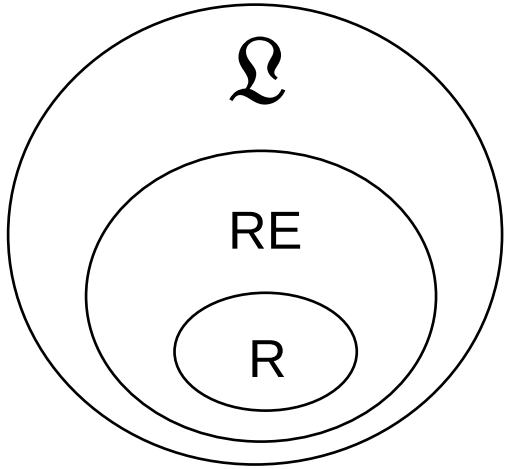


- M acepta, rechaza o no para.
- En cada iteración memoriza los estados y posiciones de las 2 ejecuciones.

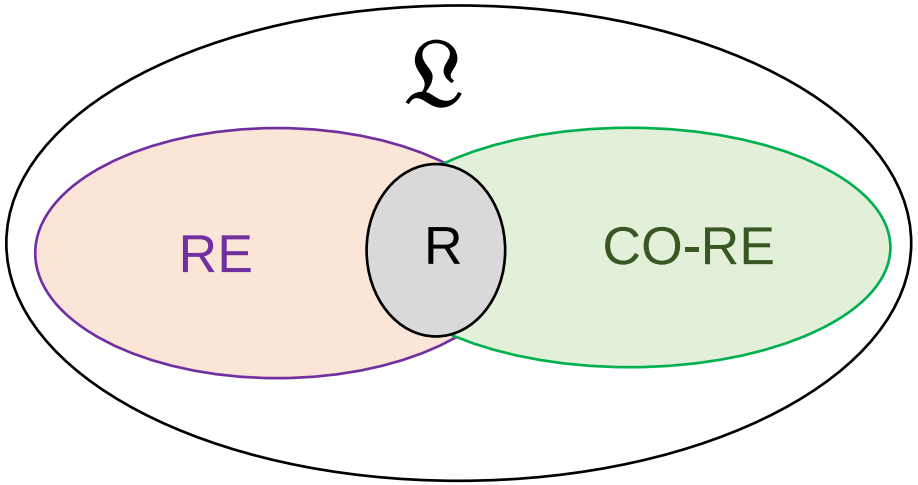
También se cumple que si $L_1 \in$ RE y $L_2 \in$ RE, entonces $L_1 \cap L_2 \in$ RE (ejercicio).

Segunda versión de la jerarquía de la computabilidad

Primera versión

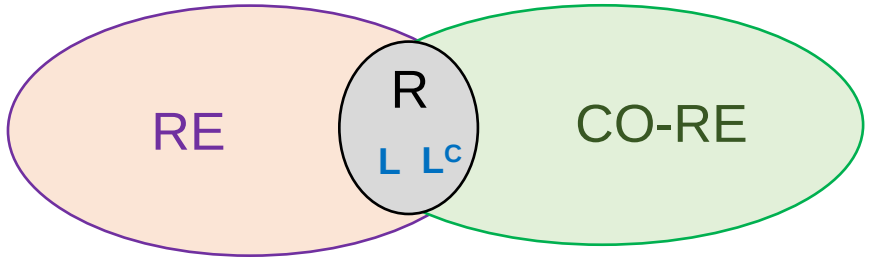


Segunda versión



El conjunto **CO-RE** tiene los complementos de los lenguajes del conjunto RE (**$L \in$ RE sii $L^C \in$ CO-RE)**

- Claramente se cumple **R $\subseteq$ RE $\cap$ CO-RE**:



R $\subseteq$ RE por definición.

R $\subseteq$ CO-RE porque:
si L $\in$ R, entonces también $L^C \in$ R, por lo que $L^C \in$ RE, y así por definición: L $\in$ CO-RE.

- Veamos que también se cumple **RE $\cap$ CO-RE $\subseteq$ R**, y por lo tanto la igualdad **R = RE $\cap$ CO-RE**.

Propiedad 4. RE ∩ CO-RE = R. Sólo falta probar la inclusión **RE ∩ CO-RE ⊆ R**:

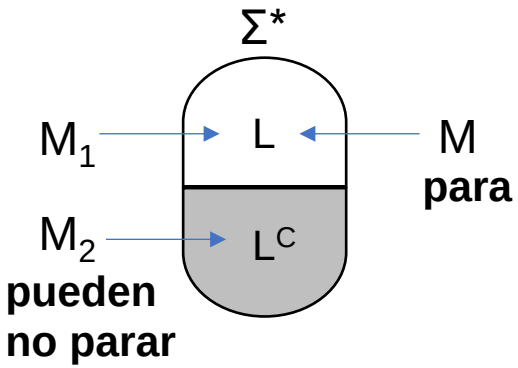
- Hay que probar:

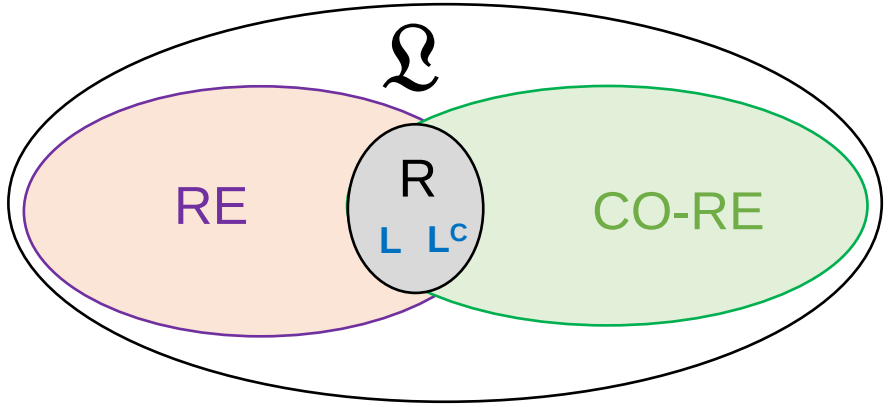
  si existe una MT M_1 que acepta L,

  y existe una MT M_2 que acepta L^C,

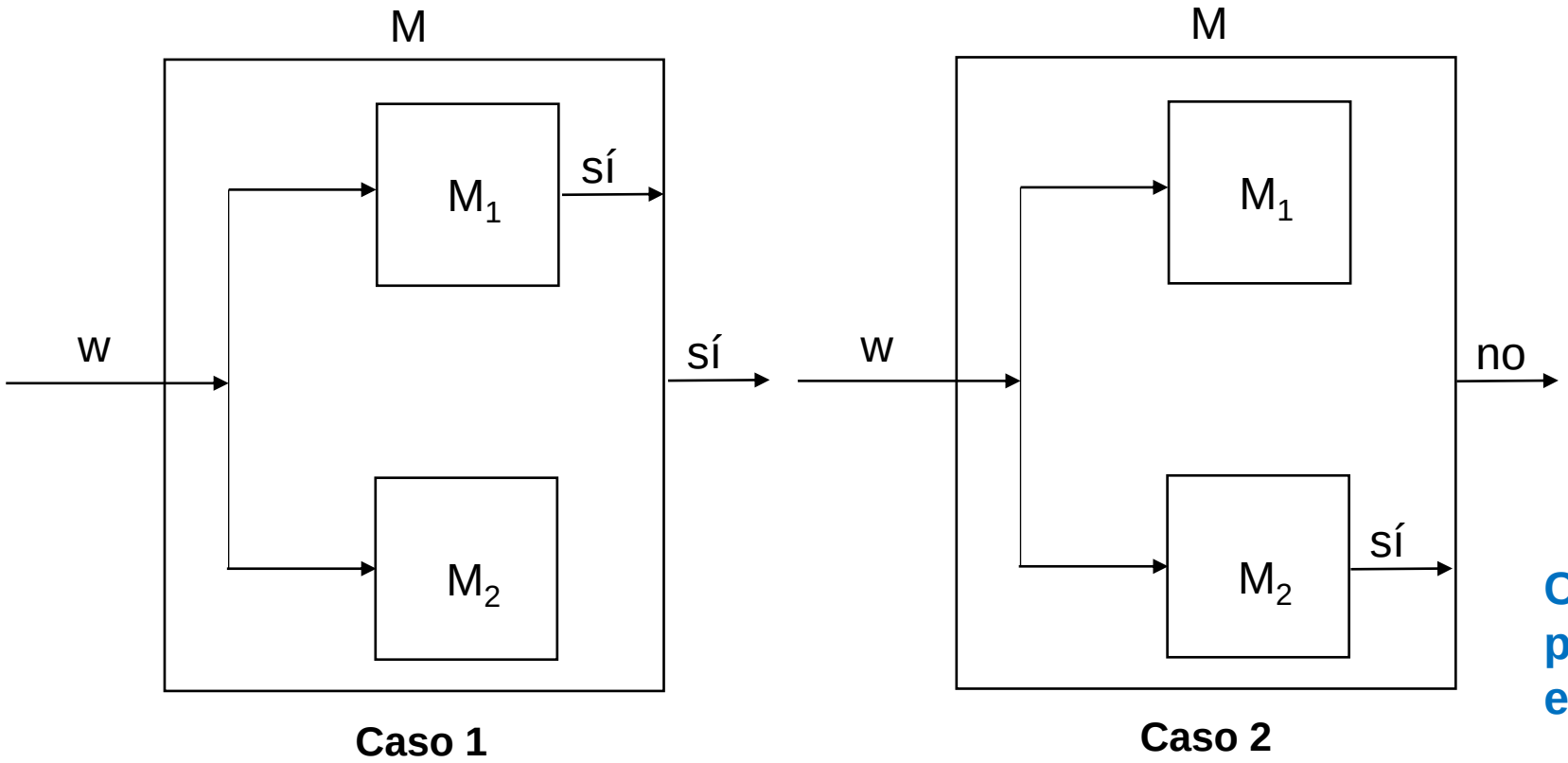
  también existe una MT M que

  acepta L y siempre para (decide L).





- Idea general:



M ejecuta en paralelo M_1 y M_2.
Si M_1 acepta w, M acepta w.
Si M_2 acepta w, M rechaza w.

Cualquiera sea w,
M_1 la acepta o M_2 la acepta.

¿Por qué?

Porque toda cadena w pertenece a L o a L^C.

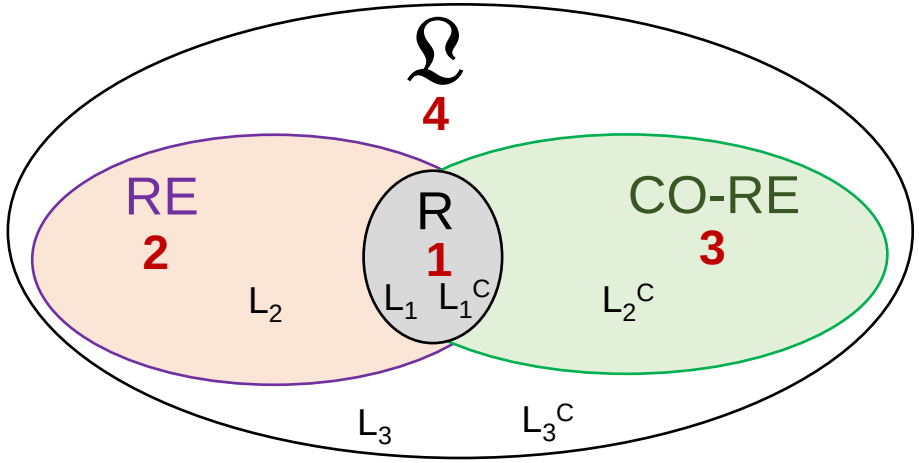
Conclusión: si un problema y el problema contrario son computables, entonces ambos son decidibles.

Las cuatro regiones de la jerarquía de la computabilidad

Región 1 (lenguajes con MT que siempre paran)
Conjunto R.
Si L está en R, entonces L^C está en R

Región 2 (lenguajes con MT)
Conjunto RE – R.
Si L está en RE, entonces L^C está en CO-RE

Región 3 (lenguajes sin MT, con complementos con MT)
Conjunto CO-RE – R.
Si L está en CO-RE, entonces L^C está en RE



Las cuatro regiones de la jerarquía, con grado de dificultad creciente.

Región 4 (lenguajes sin MT, con complementos sin MT)

Conjunto Ω – (RE U CO-RE).

Si L está en Ω – (RE U CO-RE), entonces L^C está en Ω – (RE U CO-RE)

Ejemplos clásicos de lenguajes no recursivos

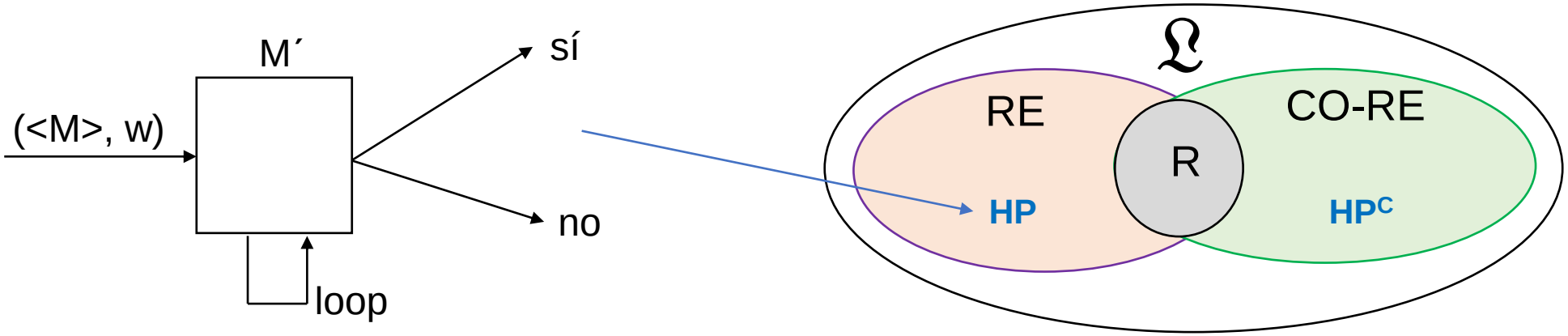
Problema de la parada de una MT o *halting problem*

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}

tal que <M> es una codificación de la MT M.

Se prueba que **HP ∈ RE – R** (A. Turing, 1936).



Ejercicio: ¿cómo funcionaría M´?

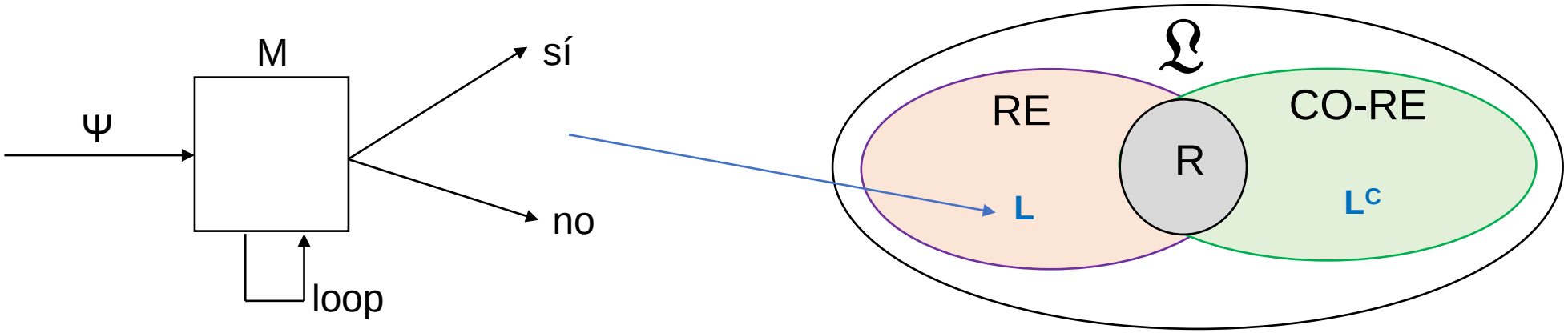
Nota: si p. ej. M sólo se mueve a la derecha, el problema es decidible.

Problema de la resolución de las ecuaciones diofánticas

Dada una ecuación algebraica con coeficientes enteros y variables enteras (*ecuación diofántica*), como por ejemplo $2x^3 + 5y^3 = 6z^3$, ¿la ecuación tiene solución? El lenguaje que representa el problema es:

L = {Ψ | Ψ es una ecuación diofántica y tiene solución}

Se prueba que **L ∈ RE – R** (Y. Matiyasevich, 1970).



Ejercicio: ¿cómo funcionaría M?

Nota: con p.ej. ecuaciones diofánticas de la forma $x^n + y^n = z^n$ con $n \geq 0$, el problema es decidible.

Problema de decisión en la lógica de predicados (LP)

¿Existe una prueba de la fórmula Φ en la LP? (¿Φ es un teorema?). El lenguaje que representa el problema es:

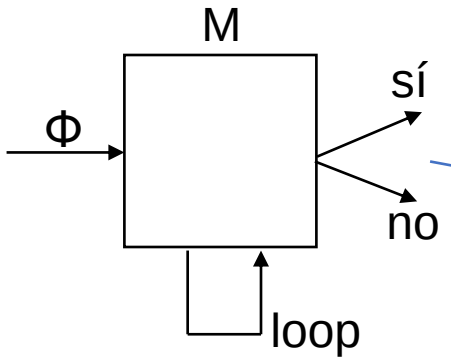
L = {Φ | existe una prueba de la fórmula Φ en la LP}

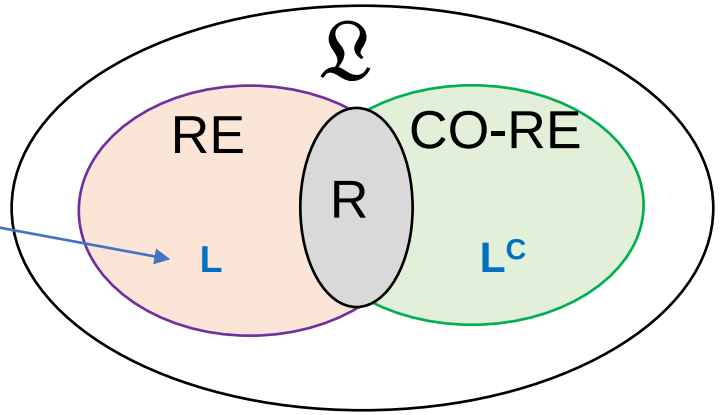
Se prueba que **L ∈ RE – R** (A. Turing, 1936).

EJEMPLO DE PRUEBA EN LP

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





Ejercicio: ¿cómo funcionaría M?

Nota: Con fórmulas de cierto tipo, el problema es decidible.

Prueba de $\Phi = \neg P \rightarrow (P \rightarrow Q)$

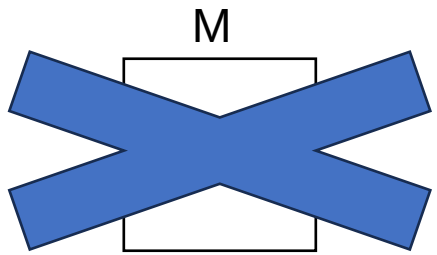
1. $\neg P \rightarrow (\neg Q \rightarrow \neg P)$ ..... axioma K_1
2. $(\neg Q \rightarrow \neg P) \rightarrow (P \rightarrow Q)$ ..... axioma K_3
3. $((\neg Q \rightarrow \neg P) \rightarrow (P \rightarrow Q)) \rightarrow (\neg P \rightarrow ((\neg Q \rightarrow \neg P) \rightarrow (P \rightarrow Q)))$ ..... 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

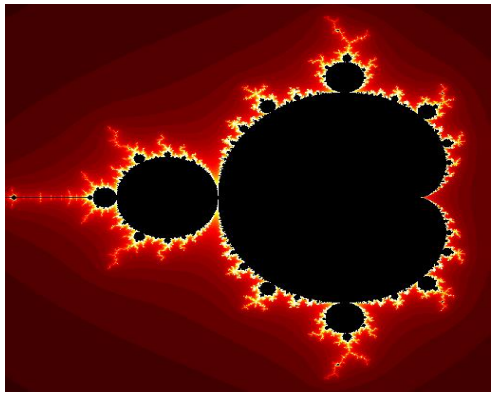
Problema de pertenencia al Conjunto de Mandelbrot (CM)

El CM es un conjunto de números complejos del plano definido por la sucesión $z_0 = 0$ y $z_{n+1} = z_n^2 + c$, tal que c está en el conjunto sii la sucesión está acotada. El problema es: dado un número complejo c, ¿c está en el CM?

El lenguaje asociado es **L = {c | c es un número complejo que está en el CM}**

Se prueba que **L ∈ CO-RE – R** (L. Blum, 1989).

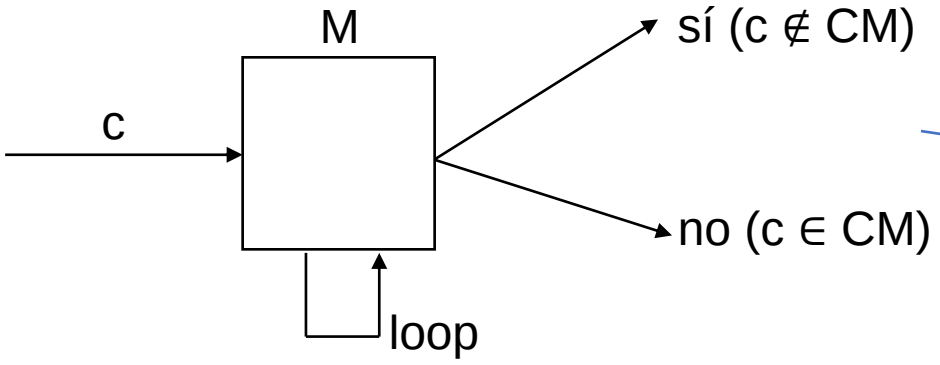


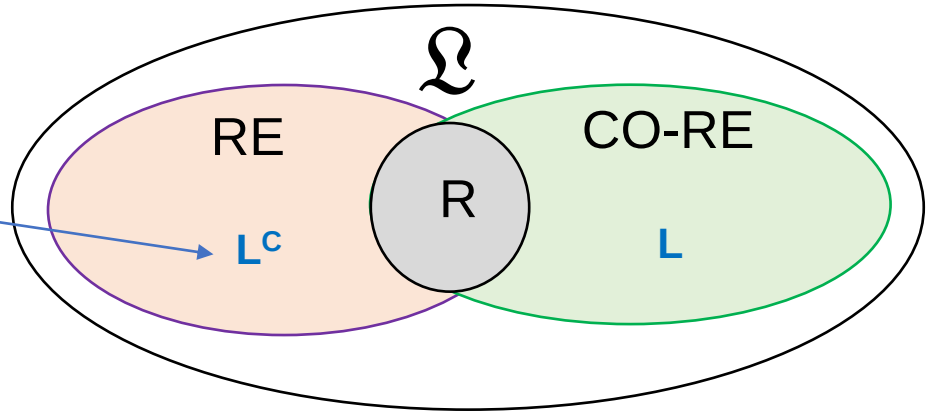


CONJUNTO DE MANDELBROT (es un fractal)

Hay elementos del CM que una MT no puede reconocer:
La dificultad está en el contorno del conjunto.

Pero al menos existe una MT M que acepta todos los números complejos que **NO** están en el CM:





Problema de decisión en la aritmética

¿La fórmula Θ es un enunciado aritmético verdadero? El lenguaje que representa el problema es:

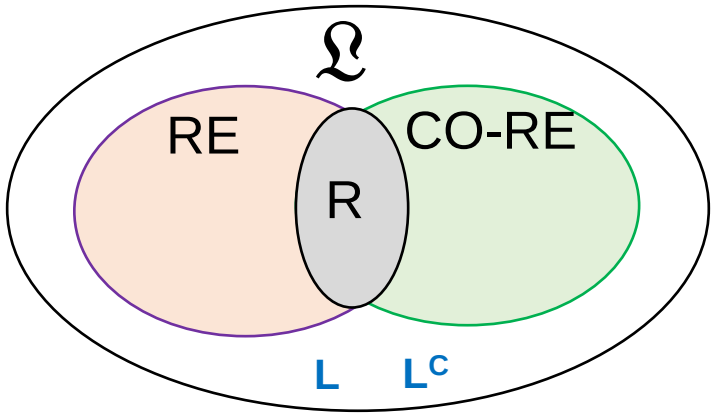
L = {Θ | Θ es un enunciado aritmético verdadero}

Se prueba que **L ∈ Ω – (RE ∪ CO-RE)** (K. Gödel, 1931).

EJEMPLO DE PRUEBA EN LA ARITMÉTICA

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

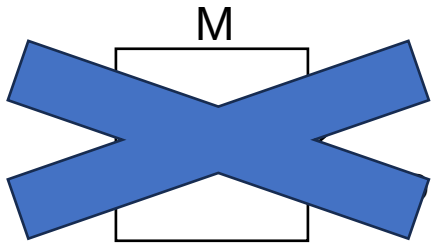


Prueba de Θ = (1 + 1 = 2) (extracto)

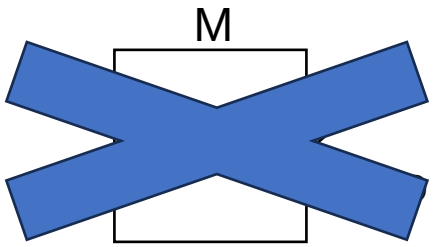
1. $(\forall x)(x + 0 = x)$
2. $(\forall x)(x + 0 = x) \rightarrow 1 + 0 = 1$
3. $1 + 0 = 1$

……

14. $s(1 + 0) = s(1) \rightarrow 1 + s(0) = s(1)$
15. $1 + s(0) = s(1)$
16. **1 + 1 = 2**



Hay enunciados verdaderos que una MT no puede reconocer.



Hay enunciados falsos que una MT no puede reconocer.

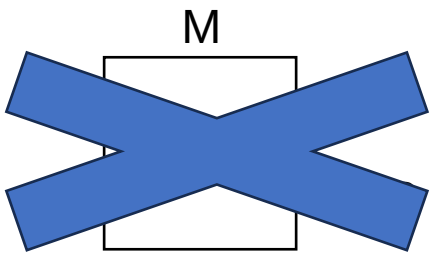
Nota: sin la multiplicación, el problema es decidible.

Problema del cubrimiento del plano con polígonos (*teselación del plano*)

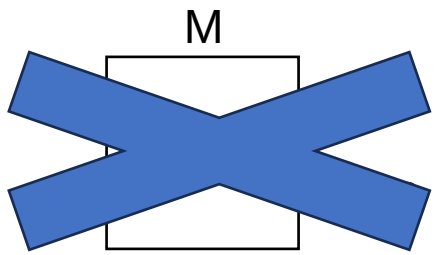
Dado un conjunto finito C de figuras poligonales (conocidas como *teselas* o *mosaicos*), ¿C puede cubrir el plano sin dejar huecos ni producir solapamientos? El lenguaje que representa el problema es:

L = {C | C es un conjunto finito de figuras poligonales que cubren el plano}

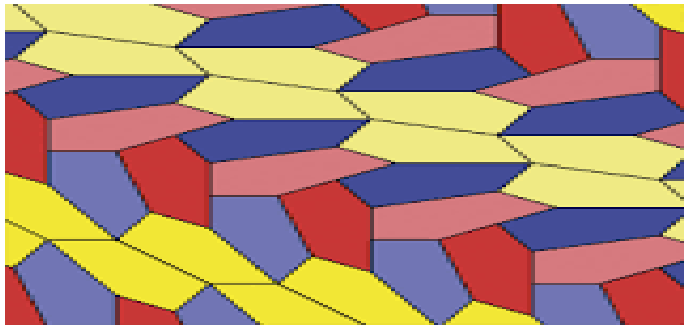
Se prueba que **L ∈ Ω – (RE** ∪ **CO-RE)** (R. Berger, 1966).



Hay conjuntos de polígonos que cubren el plano que una MT no puede reconocer.

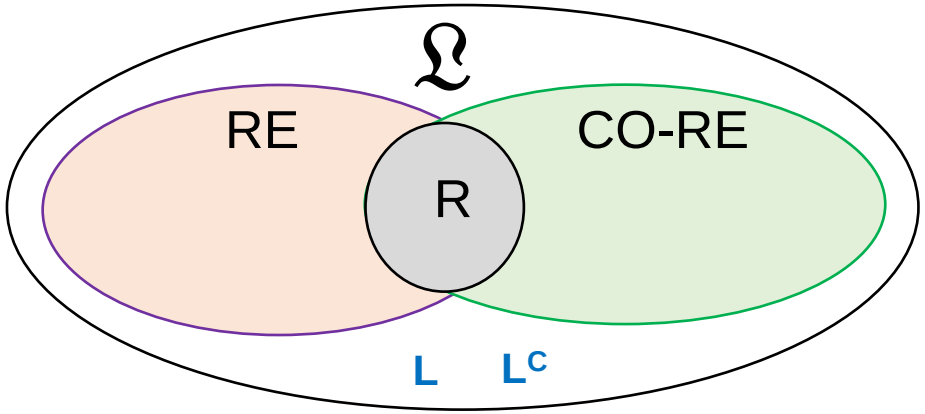


Hay conjuntos de polígonos que no cubren el plano que una MT no puede reconocer.



EJEMPLO DE TESELACIÓN

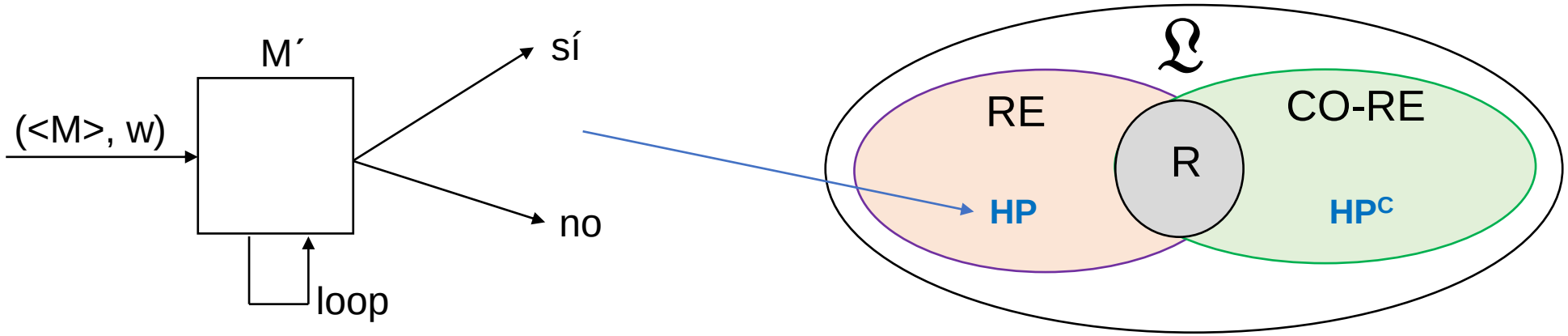
Nota: si hay periodicidad, el problema es decidible.



Anexo

Problema del *halting problem* revisitado

HP = {(<M>, w) | M para a partir de w} ∈ RE – R



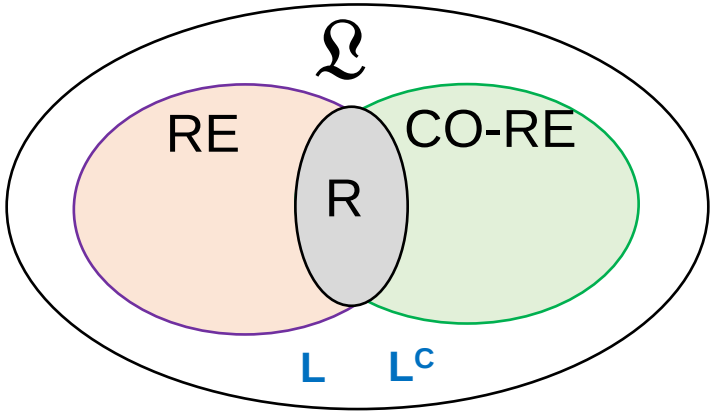
- No existe ninguna MT M´ que para **toda** MT M y **toda** cadena w, pueda decidir si M para a partir de w.
- Pero para **algunas** MT M y **algunas** cadenas w, existen MT M´ que **SÍ** pueden decidir el problema.
- Por ejemplo, si M va **siempre a la derecha**, si M se mueve en un **espacio acotado de celdas**, etc., **se puede decidir** (mediante MT M´ “ingeniosas”).
- En general se considera el **peor caso**, es decir, todas las entradas posibles. En dicho caso, el único algoritmo posible para el *halting problem* es la **fuerza bruta** (ejecutar M y esperar a que eventualmente responda).

Problema de decisión en la aritmética revisitado

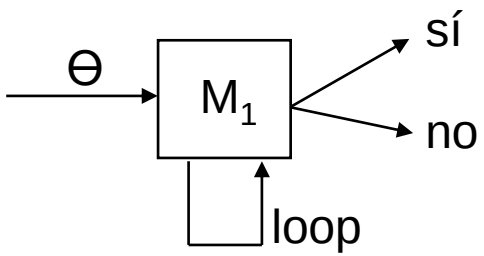
Teorema de Incompletitud (K. Gödel, 1931):
Existen enunciados aritméticos verdaderos que no se pueden probar.

Es decir, L = {Θ | Θ es un enunciado aritmético verdadero} ∉ RE.

Veamos que también L^C = {Θ | Θ es un enunciado aritmético falso} ∉ RE:

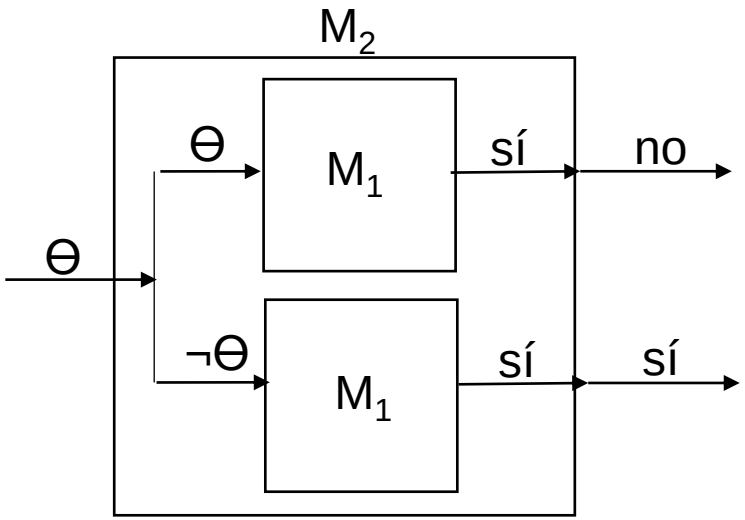


Supongamos lo contrario, que L^C ∈ RE. Así, existe la siguiente MT M_1:



M acepta sii Θ es falso

De esta manera, también existe la siguiente MT M_2:

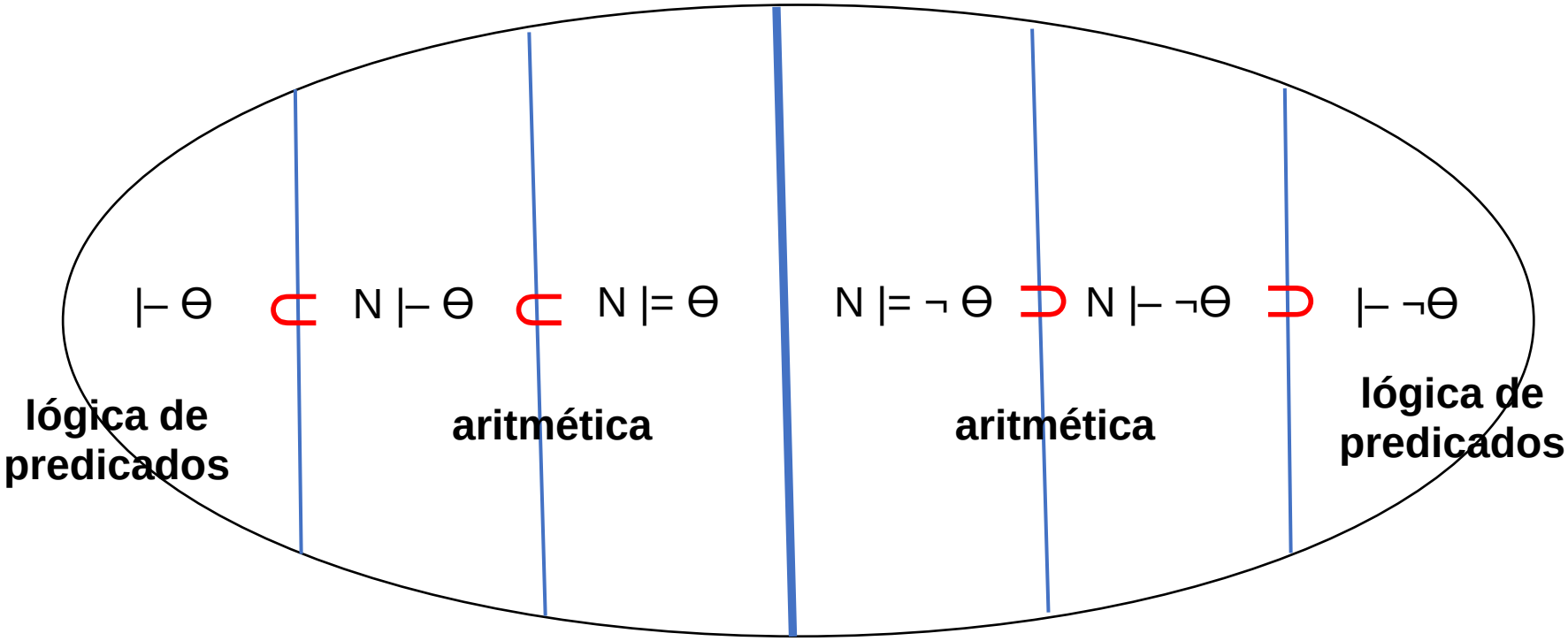


Se cumple:

- **Θ es verdadero o falso**.
- **Si Θ es verdadero, M_2 acepta.**
- **Si Θ es falso, M_2 rechaza.**

En consecuencia, M_2 acepta L y para siempre, lo que significa que L es recursivo (**absurdo, y así no puede suceder que L^C ∈ RE**).

Indecibilidad en la lógica de predicados y en la aritmética



- |– Θ es el conjunto de los teoremas de la lógica de predicados. **Pertenece a RE – R.**
- N |– Θ es el conjunto de los teoremas de la aritmética. **Pertenece a RE – R.**
- N |= Θ es el conjunto de los enunciados verdaderos de la aritmética. **No pertenece a RE.**

Clase práctica 2

Ejercicio 1. Probar que la clase R es cerrada con respecto a la operación de concatenación.
Es decir: 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 $w = v_1v_2$, tales que la subcadena $v_1 \in L_1$ y la subcadena $v_2 \in L_2$.

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 acepte $L_1 . L_2$ ejecutando sobre un input w (de n símbolos) determinadas MT M_1 y M_2 (MT que aceptan 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 $M_1 = (Q_1, \Gamma_1, \delta_1, q'_0, q_A, q_R)$ y $M_2 = (Q_2, \Gamma_2, \delta_2, q''_0, q_A, q_R)$.

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:

$M = (Q = Q_1 \cup Q_2 \cup \{q_0\}, \Gamma = \Gamma_1 \cup \Gamma_2, \Delta, q_0, q_A, q_R)$, 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 Γ.

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.