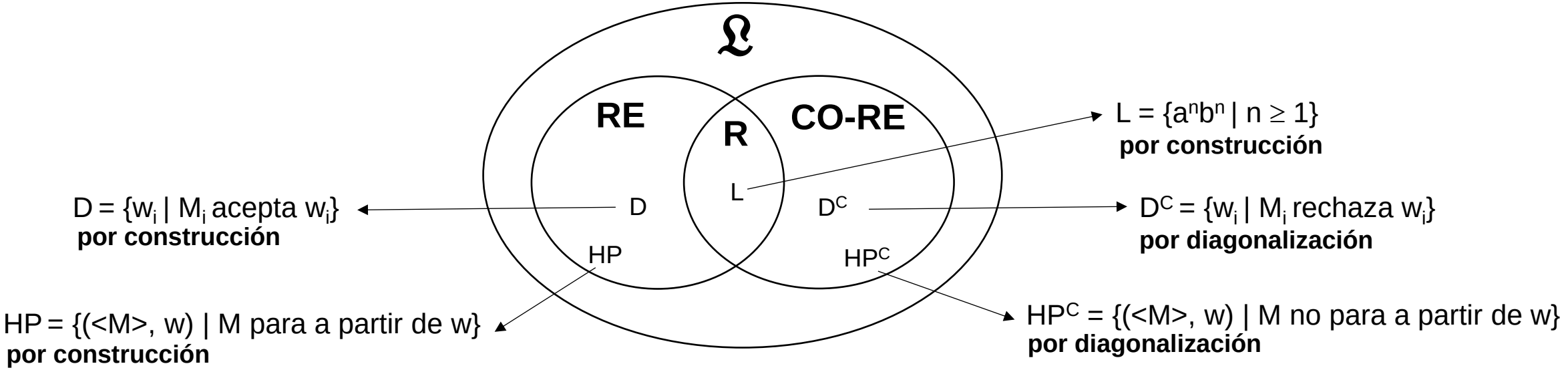
Clase teórica 4

Las reducciones
y
otros temas de computabilidad

Repaso

- Hemos probado cómo se ubican algunos primeros lenguajes dentro de la jerarquía de la computabilidad:



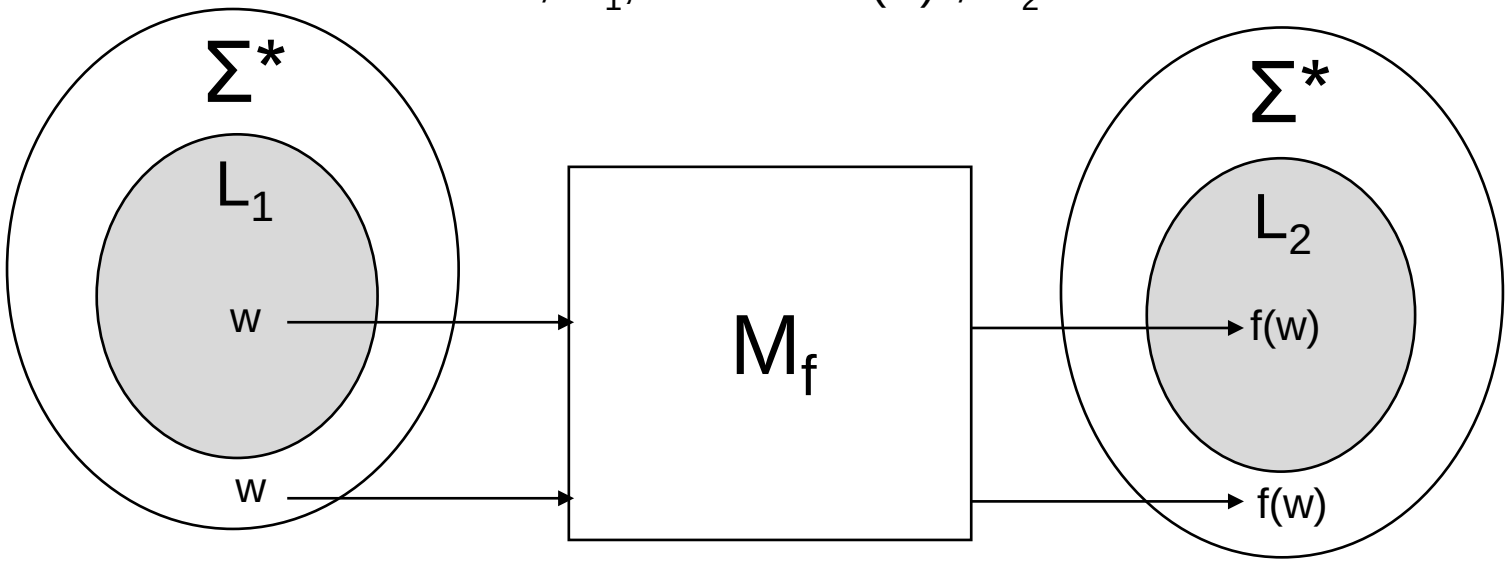
- Para probar **pertenencia a R y RE** hemos **construido MT.**
- Para probar **no pertenencia a R y RE** hemos utilizado el método de **diagonalización.**
- En esta clase, que cierra la parte de computabilidad, vamos a estudiar otro método para probar **no pertenencia a R y RE**, en general más sencillo que la diagonalización: la **reducción**.

Reducciones de lenguajes

- Dados dos lenguajes L_1 y L_2, **una reducción de L_1 a L_2** es una **función total computable** $f : \Sigma^* \longrightarrow \Sigma^*$ (función definida para todas las cadenas de Σ^* y computable por una MT M_f), que para toda cadena w:

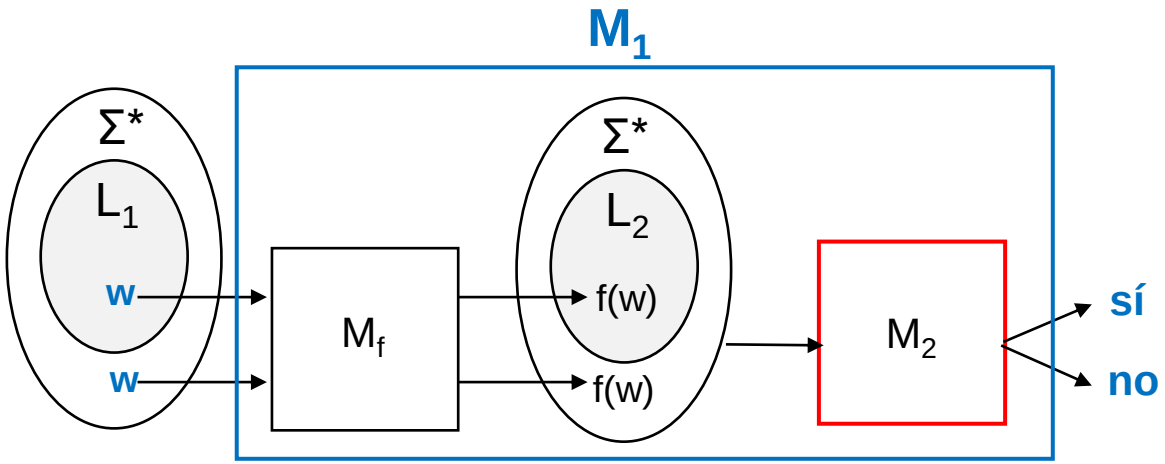
si $w \in L_1$, entonces $f(w) \in L_2$

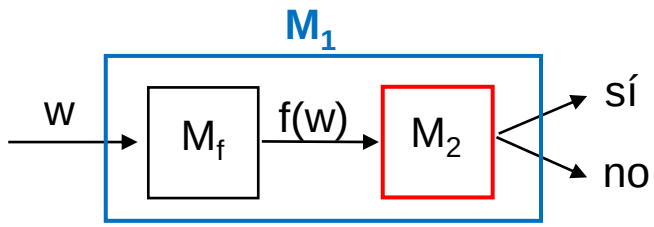
si $w \notin L_1$, entonces $f(w) \notin L_2$



- **Objetivo:** reducir (relacionar) el lenguaje L_1 al lenguaje L_2, con la idea de construir una MT M_1 que acepte L_1 utilizando una MT M_2 ya conocida que acepta L_2, **en lugar de construir M_1 de cero**.
- ¿A qué técnica de programación se asemeja esto? **A la invocación a una subrutina.**
- La notación $\mathbf{L_1 \leq L_2}$ expresa que existe una reducción de L_1 a L_2 .

Formalizando la utilidad de las reducciones (caso 1: $L_2 \in R$)





Si M_2 decide L_2,
entonces M_1 decide L_1.

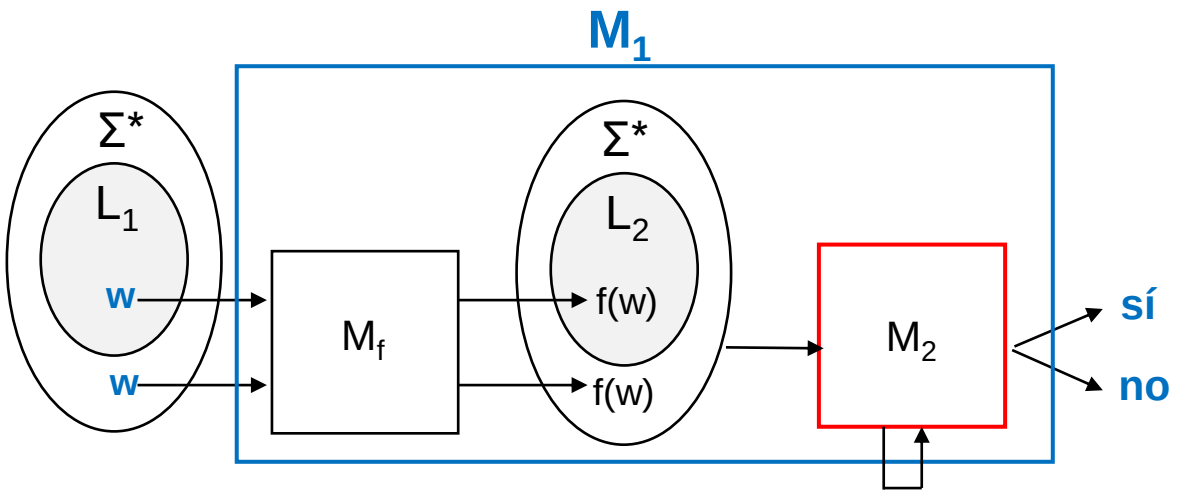
Si $w \in L_1$, entonces $f(w) \in L_2$, entonces M_2 acepta f(w), entonces M_1 acepta w.
Si $w \notin L_1$, entonces $f(w) \notin L_2$, entonces M_2 rechaza f(w), entonces M_1 rechaza w.

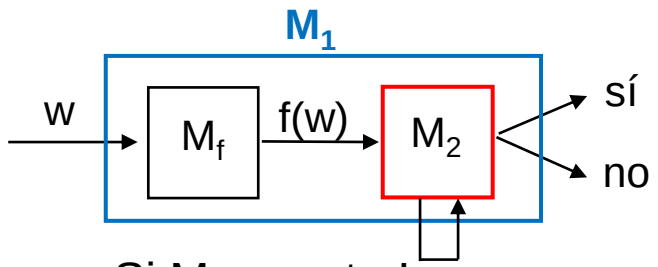
- En lugar de construir de cero una MT M_1 para decidir L_1, se la puede construir a partir de una MT M_2 **conocida** que decide L_2 (noción de invocación a una subrutina en programación).

- Formalmente: **si $L_1 \leq L_2$, entonces $L_2 \in R \longrightarrow L_1 \in R$**
  O lo mismo: **si $L_1 \leq L_2$, entonces $L_1 \notin R \longrightarrow L_2 \notin R$**

Encontrando una reducción de un lenguaje L_1 que no está en R a un lenguaje L_2, se prueba que L_2 tampoco está en R.

Formalizando la utilidad de las reducciones (caso 2: $L_2 \in RE$)





Si M_2 acepta L_2,
entonces M_1 acepta L_1.

Si $w \in L_1$, entonces $f(w) \in L_2$, entonces M_2 acepta f(w), entonces M_1 acepta w.
Si $w \notin L_1$, entonces $f(w) \notin L_2$, entonces M_2 rechaza f(w) (puede loopear), entonces M_1 rechaza w (puede loopear).

- En lugar de construir de cero una MT M_1 para aceptar L_1, se la puede construir a partir de una MT M_2 **conocida** que acepta L_2 (noción de invocación a una subrutina en programación).

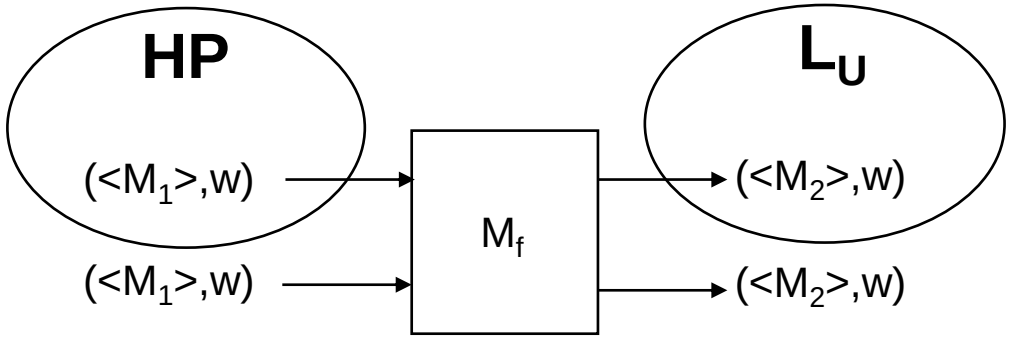
- Formalmente: **si $L_1 \leq L_2$, entonces $L_2 \in RE \longrightarrow L_1 \in RE$**
  O lo mismo: **si $L_1 \leq L_2$, entonces $L_1 \notin RE \longrightarrow L_2 \notin RE$**

Encontrando una reducción de un lenguaje L_1 que no está en RE a un lenguaje L_2, se prueba que L_2 tampoco está en RE.

Ejemplo 1

HP = {(<M>, w) | M para a partir de w} y L_U = {(<M>, w) | M acepta w}.

Vamos a probar **HP ≤ L_U**. Idea general:



Sabemos que: si $L_1 \leq L_2$, entonces $L_1 \notin R \longrightarrow L_2 \notin R$

Así, de HP ≤ L_U y HP ∉ R, probamos **$L_U \notin R$**

Reducción (gráfico): M_2 es como M_1, salvo que los estados q_R de M_1 se cambian en M_2 por estados q_A.

Computabilidad: M_f copia (<M_1>, w) pero cambiando los estados q_R de M_1 por estados q_A en M_2.

Correctitud:

(<M_1>, w) ∈ HP ⟶ M_1 para desde w ⟶ M_2 acepta w (¿por qué?) ⟶ (<M_2>, w) ∈ L_U

(<M_1>, w) ∉ HP ⟶ si (<M_1>, w) es una cadena válida:

M_1 no para desde w ⟶ M_2 no para desde w (¿por qué?) ⟶ (<M_2>, w) ∉ L_U

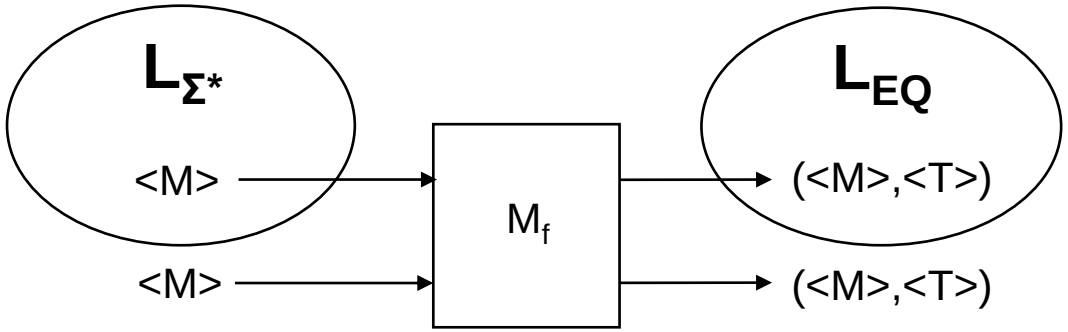
si (<M_1>, w) no es una cadena válida:

(<M_2 >, w) tampoco es una cadena válida ⟶ (<M_2>, w) ∉ L_U

Ejemplo 2

$L_{\Sigma^*} = \{<M> \mid L(M) = \Sigma^*\}$ y $L_{EQ} = \{(<M_1>, <M_2>) \mid L(M_1) = L(M_2)\}$.

Vamos a probar $\mathbf{L_{\Sigma^*} \leq L_{EQ}}$. Idea general (*comentario: se prueba que $L_{\Sigma^*} \notin RE$*):



Sabemos que: Si $L_1 \leq L_2$, entonces $L_1 \notin RE \rightarrow L_2 \notin RE$

Así, de $L_{\Sigma^*} \leq L_{EQ}$ y $L_{\Sigma^*} \notin RE$, probamos **$L_{EQ} \notin RE$**

Reducción (gráfico): <T> es una MT que acepta el lenguaje Σ^* (¿cómo sería la MT T?)

Computabilidad: M_f copia <M> y le concatena <T>.

Correctitud:

$<M> \in L_{\Sigma^*} \rightarrow L(M) = \Sigma^* \rightarrow L(M) = L(T) \rightarrow (<M>, <T>) \in L_{EQ}$

$<M> \notin L_{\Sigma^*} \rightarrow$ si <M> es una cadena válida:

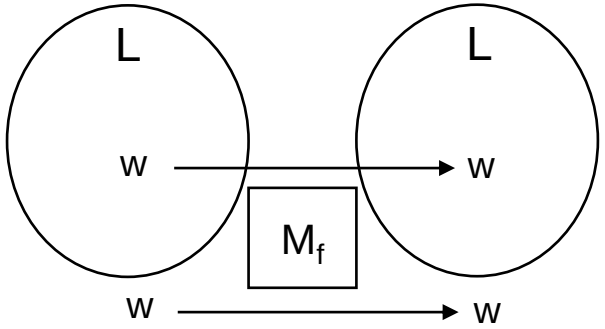
$L(M) \neq \Sigma^* \rightarrow L(M) \neq L(T) \rightarrow (<M>, <T>) \notin L_{EQ}$

si <M> no es una cadena válida:

(<M>, <T>) tampoco es una cadena válida $\rightarrow (<M>, <T>) \notin L_{EQ}$

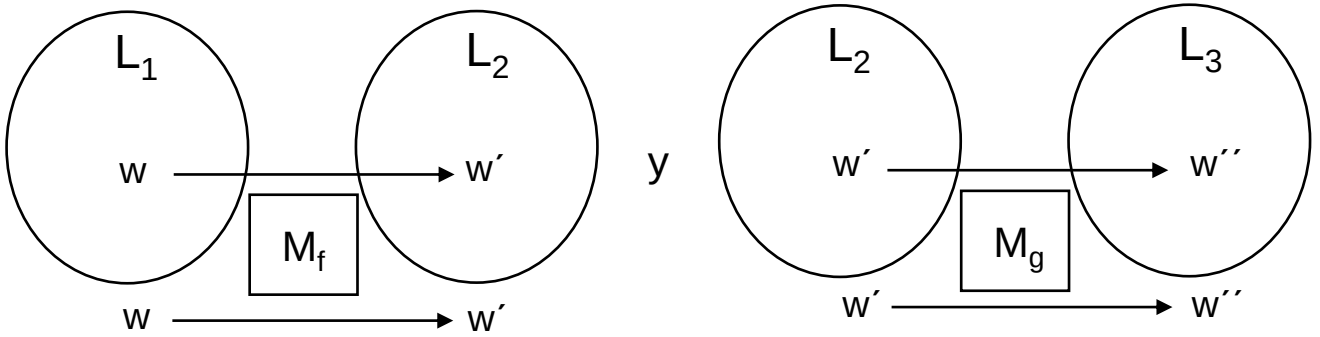
Algunas propiedades de las reducciones

- **Reflexividad.** Para todo lenguaje L se cumple $L \leq L$.
  La reducción es la función **identidad**.



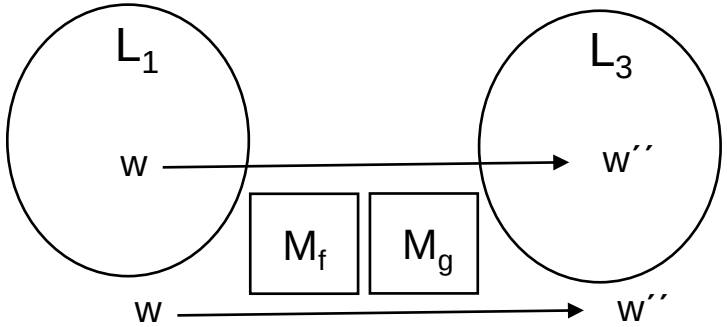
reducción f(w) = w

- **Transitividad.** Si $L_1 \leq L_2$ y $L_2 \leq L_3$, entonces $L_1 \leq L_3$.



y

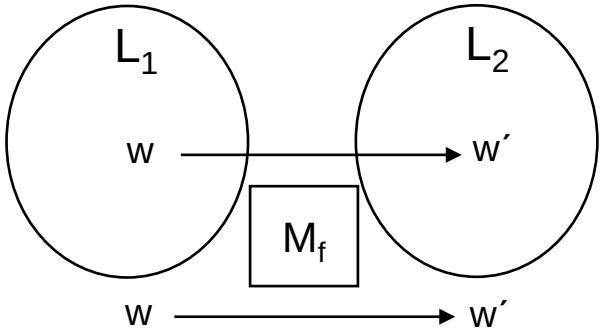
entonces



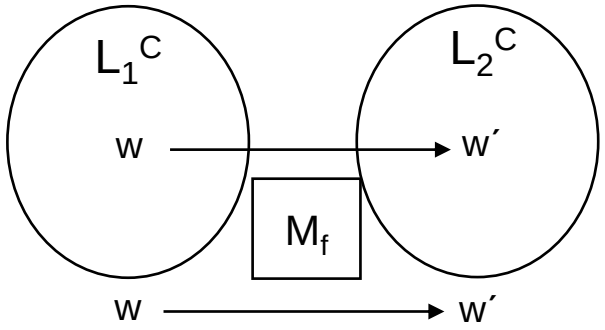
La reducción es la **composición** de las dos reducciones.

reducción g(f(w))

- **Otra propiedad:** $L_1 \leq L_2$ sii $L_1^C \leq L_2^C$. La reducción es la **misma**.

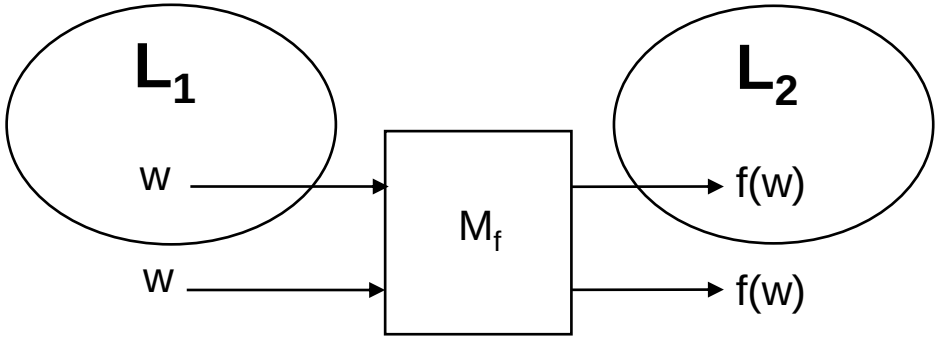


sii



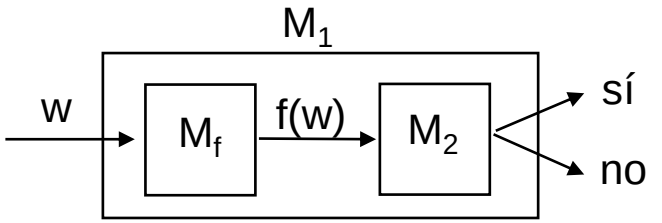
No se cumple la simetría.
$L_1 \leq L_2$ no implica $L_2 \leq L_1$.

Resumiendo



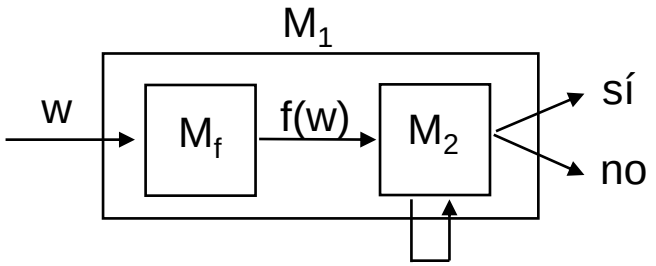
Caso 1

Si $L_1 \leq L_2$ entonces ($L_2 \in R \longrightarrow L_1 \in R$)
O bien, por el contrarrecíproco:
Si $L_1 \leq L_2$ entonces ($L_1 \notin R \longrightarrow L_2 \notin R$)



Caso 2

Si $L_1 \leq L_2$ entonces ($L_2 \in RE \longrightarrow L_1 \in RE$)
O bien, por el contrarrecíproco:
Si $L_1 \leq L_2$ entonces ($L_1 \notin RE \longrightarrow L_2 \notin RE$)

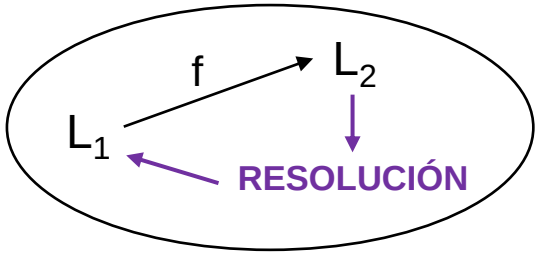


- Expresado de otra manera:

Si $L_1 \leq L_2$, entonces L_2 es tan o más difícil que L_1

Si $L_1 \notin R$, no puede suceder que $L_2 \in R$

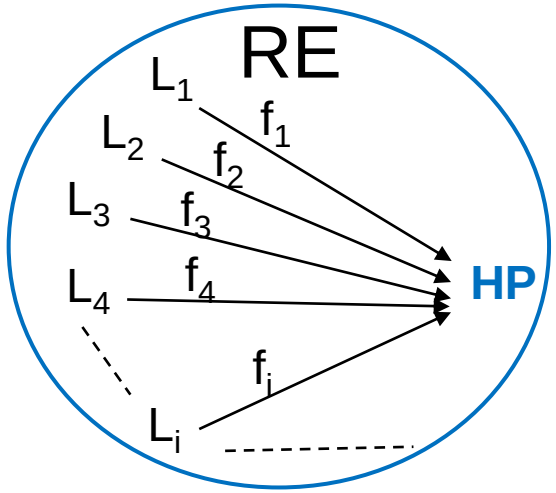
Si $L_1 \notin RE$, no puede suceder que $L_2 \in RE$



Resolviendo L_2 se resuelve L_1

Otra manera de probar que HP está entre los lenguajes más difíciles de RE

- En la clase anterior lo vimos construyendo una MT.
- Lo mismo se puede ver por medio de las reducciones. **Se prueba que todo lenguaje L de RE cumple L ≤ HP:**



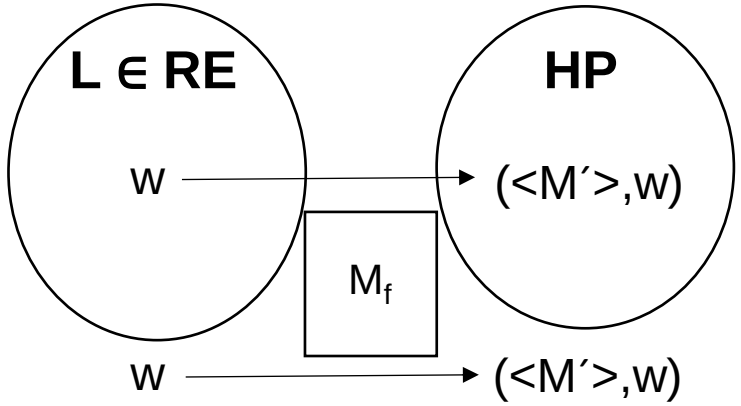
HP es tan o más difícil que cualquier lenguaje de RE.

Decidiendo HP se decide cualquier lenguaje L_i.

Así, si HP estuviera en R, se cumpliría R = RE.

HP identifica la dificultad de la clase RE, es RE-completo.

Idea general de la prueba



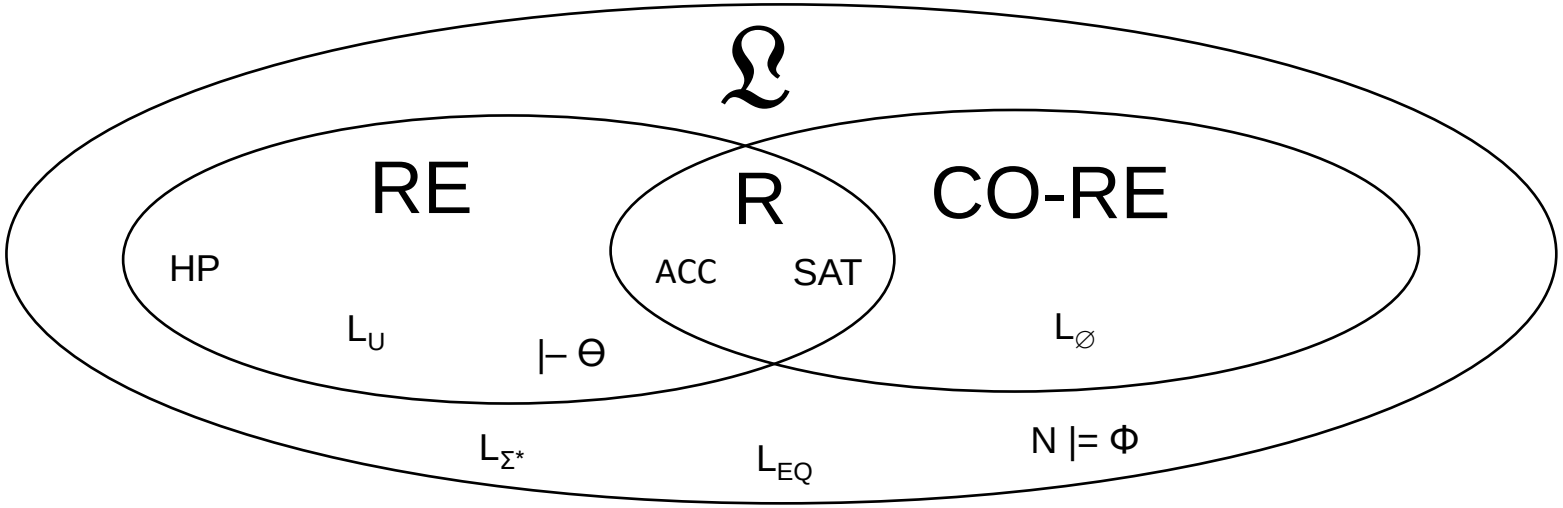
Sea M una MT que acepta L.

M´ es como M, salvo que desde todo q_R de M se incluye en M´ un loop.

De esta manera:

w ∈ L⟶ M acepta w ⟶ M´ para a partir de w.
w ∉ L⟶ M rechaza w ⟶ M´ no para a partir de w.

Una última mirada a problemas clásicos de la computabilidad

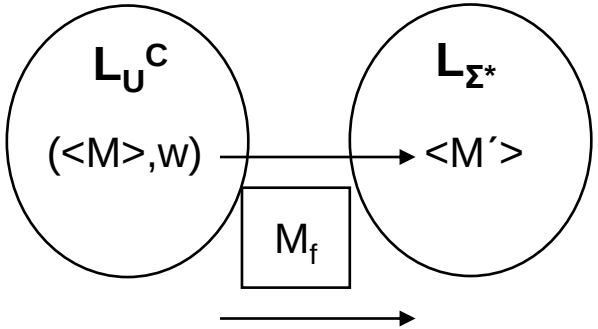


- ACC, **problema de accesibilidad**: ¿El grafo no dirigido G tiene un camino de su primer a su último vértice?
- SAT, **problema de satisfactibilidad**: ¿La fórmula booleana φ es satisfactible?
- HP, **problema de la parada**: ¿La MT M para a partir de la cadena de entrada w?
- L_U, **problema de aceptación universal**: ¿La MT M acepta la cadena de entrada w?
- |– Θ, **problema de decisión en la lógica de predicados (LP):** ¿La fórmula Θ es un teorema de la LP?
- $L_\varnothing$: ¿El lenguaje aceptado por la MT M es el lenguaje vacío?
- L_{Σ^*}: ¿El lenguaje aceptado por la MT M es el lenguaje Σ*?
- L_{EQ}, **problema de equivalencia**: ¿Las MT M y M´ son equivalentes?
- N |= Φ, **problema de decisión en la aritmética**: ¿El enunciado aritmético Φ es verdadero?

Anexo

Otros ejemplos clásicos de reducciones

1) $L_U^C \leq L_{\Sigma^*}$

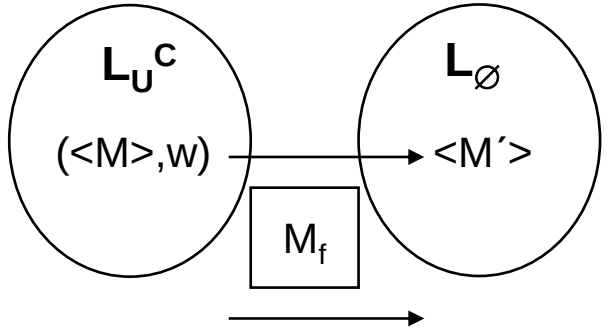


Como $L_U^C \notin$ RE, entonces **$L_{\Sigma^*} \notin$ RE.**

También se cumple $L_U \leq L_{\Sigma^*}$ (se prueba en la clase práctica).
Como $L_1 \leq L_2$ sii $L_1^C \leq L_2^C$, entonces $L_U^C \leq L_{\Sigma^*}^C$, y así **$L_{\Sigma^*}^C \notin$ RE.**

En definitiva: **L_{Σ^*} y $L_{\Sigma^*}^C$ están en $\mathcal{L}$ – (RE ∪ CO-RE).**

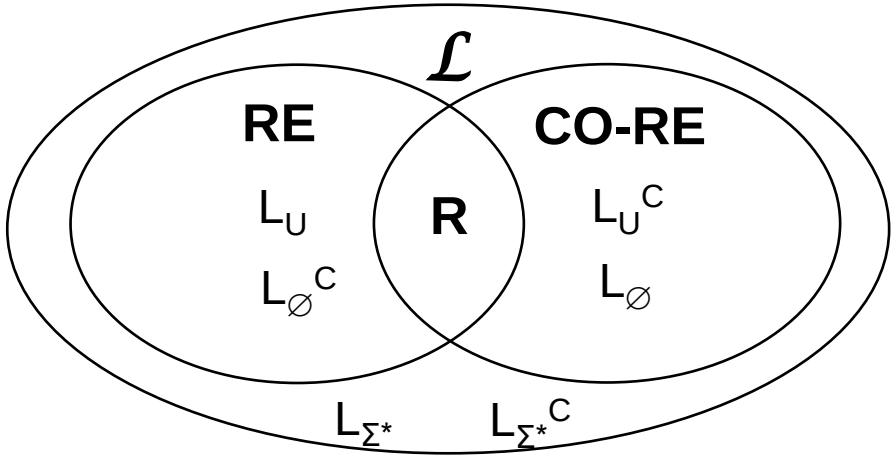
2) $L_U^C \leq L_\varnothing$



Como $L_U^C \notin$ RE, entonces **$L_\varnothing \notin$ RE.**

Por otro lado, probamos antes, por construcción, que **$L_\varnothing^C \in$ RE**
(se puede detectar si una MT acepta al menos una cadena).

En definitiva: **$L_\varnothing$ está en CO-RE y $L_\varnothing^C$ está en RE.**



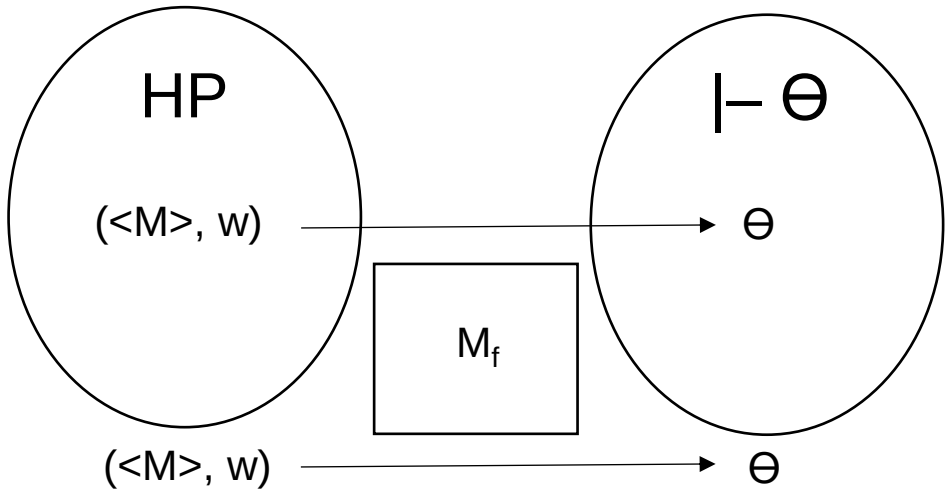
$L_U = \{(<M>, w) \mid M \text{ acepta } w\}$.

$L_{\Sigma^*} = \{<M> \mid L(M) = \Sigma^*\}$

$L_\varnothing = \{<M> \mid L(M) = \varnothing\}$

Prueba de indecibilidad en la lógica de predicados (Turing, 1936)

- Reducción de HP a |– Θ



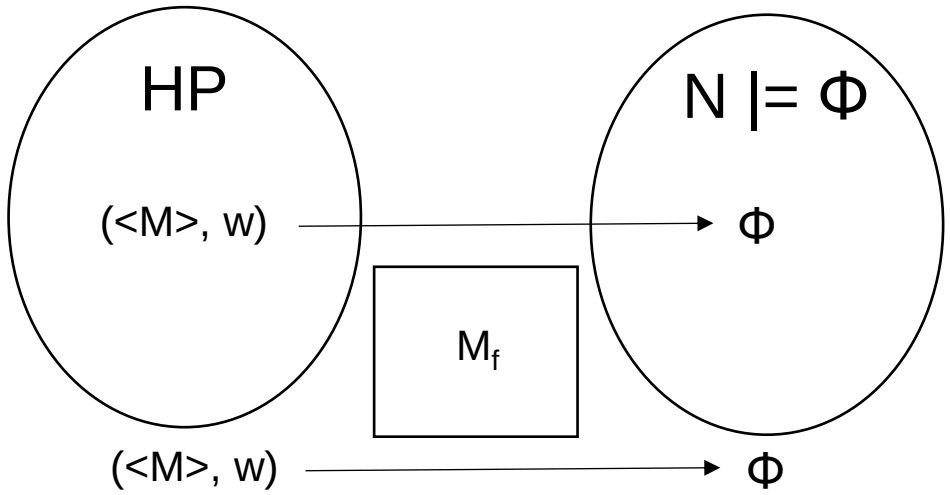
M para a partir de w sii Θ es una fórmula demostrable en la lógica de predicados.

Sabemos que $L_1 \leq L_2$ implica ($L_1 \notin R \longrightarrow L_2 \notin R$). Por lo tanto, **como HP ∉ R, entonces |– Θ ∉ R.**

La lógica de predicados es indecidible, porque el *halting problem* es indecidible.

Prueba de indecibilidad en la aritmética (alternativa a la prueba de Gödel de 1931)

- <u>Reducción de HP a N |= Φ</u>



M para a partir de w sii Φ es un enunciado verdadero de la aritmética.

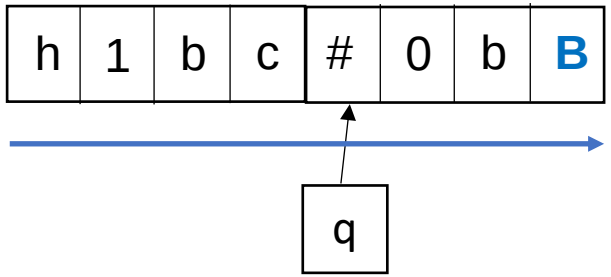
Sabemos que $L_1 \leq L_2$ implica ($L_1 \notin R \longrightarrow L_2 \notin R$). Por lo tanto, **como HP ∉ R, entonces N |= Φ ∉ R.**

La aritmética es indecidible, porque el *halting problem* es indecidible.

Máquinas de Turing restringidas

AUTÓMATAS FINITOS (AF)

- Una cinta de sólo lectura.
- Sólo movimiento a la derecha.
- Conjunto F de estados finales.



- Cuando alcanza el símbolo B (blanco), el AF para, y acepta sii el estado alcanzado es un estado final.

- El AF constituye un tipo de algoritmo muy utilizado. Por ejemplo:
  - Para el **análisis sintáctico a nivel palabra** de los compiladores (if, then, else, while, x, 10, =, +, etc).
  - Para las **inspecciones de código** en el control de calidad del software.

Ejemplo

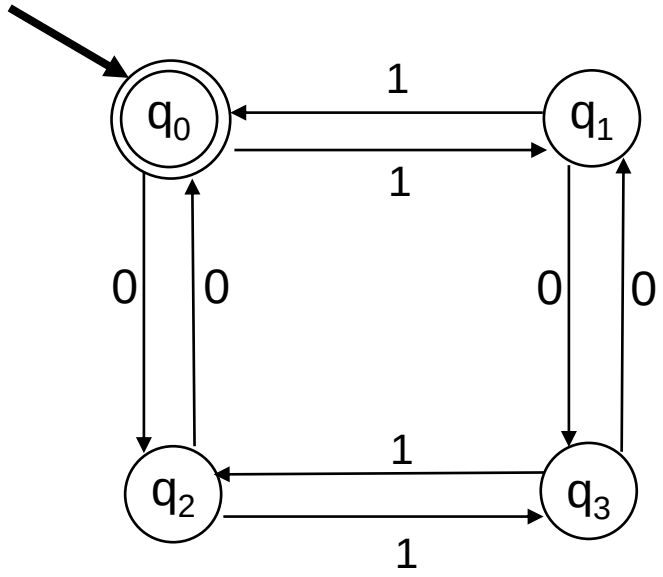
$Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, Estado inicial q_0, $F = \{q_0\}$

1. $\delta(q_0, 1) = q_1$
2. $\delta(q_1, 1) = q_0$
3. $\delta(q_1, 0) = q_3$
4. $\delta(q_3, 0) = q_1$
5. $\delta(q_3, 1) = q_2$
6. $\delta(q_2, 1) = q_3$
7. $\delta(q_2, 0) = q_0$
8. $\delta(q_0, 0) = q_2$

Diagrama de transición de estados

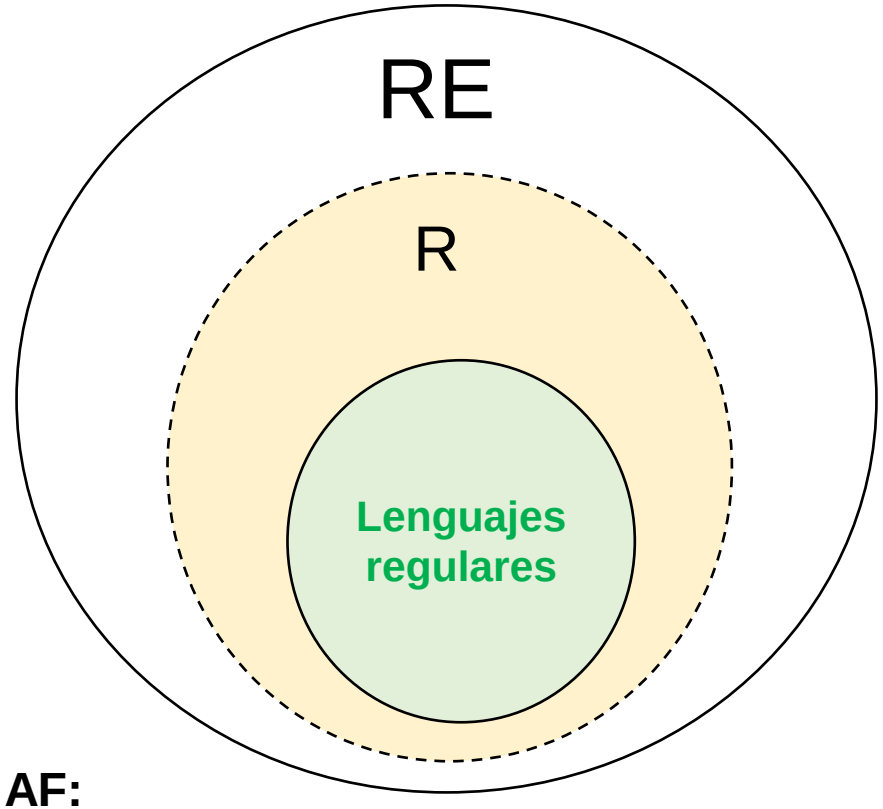
La ejecución arranca desde la flecha.

Los estados con doble contorno son los estados finales.



El AF descripto acepta todas las cadenas de 1 y 0 **con una cantidad par de 1 y una cantidad par de 0.**

Algunas características de los AF



- Los AF **siempre paran.**

- Aceptan un tipo limitado de cadenas **(no tienen memoria).**

  No pueden aceptar cadenas con igual cantidad de *a* y b,
  no pueden chequear si una cadena es un palíndromo, etc.
  En general, **no pueden calcular**.

- Los lenguajes que aceptan los AF se llaman **regulares** o de **tipo 3**.

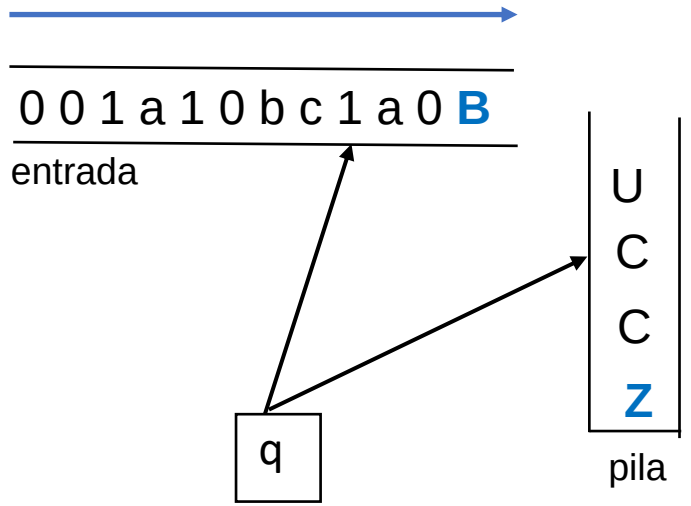
- A diferencia de las MT generales, **siempre se puede decidir si un AF:**

  Acepta una cadena w.
  Acepta el lenguaje ∅.
  Acepta el lenguaje Σ*.
  Es equivalente a otro.
  Etc.

AUTÓMATAS CON PILA (AP)

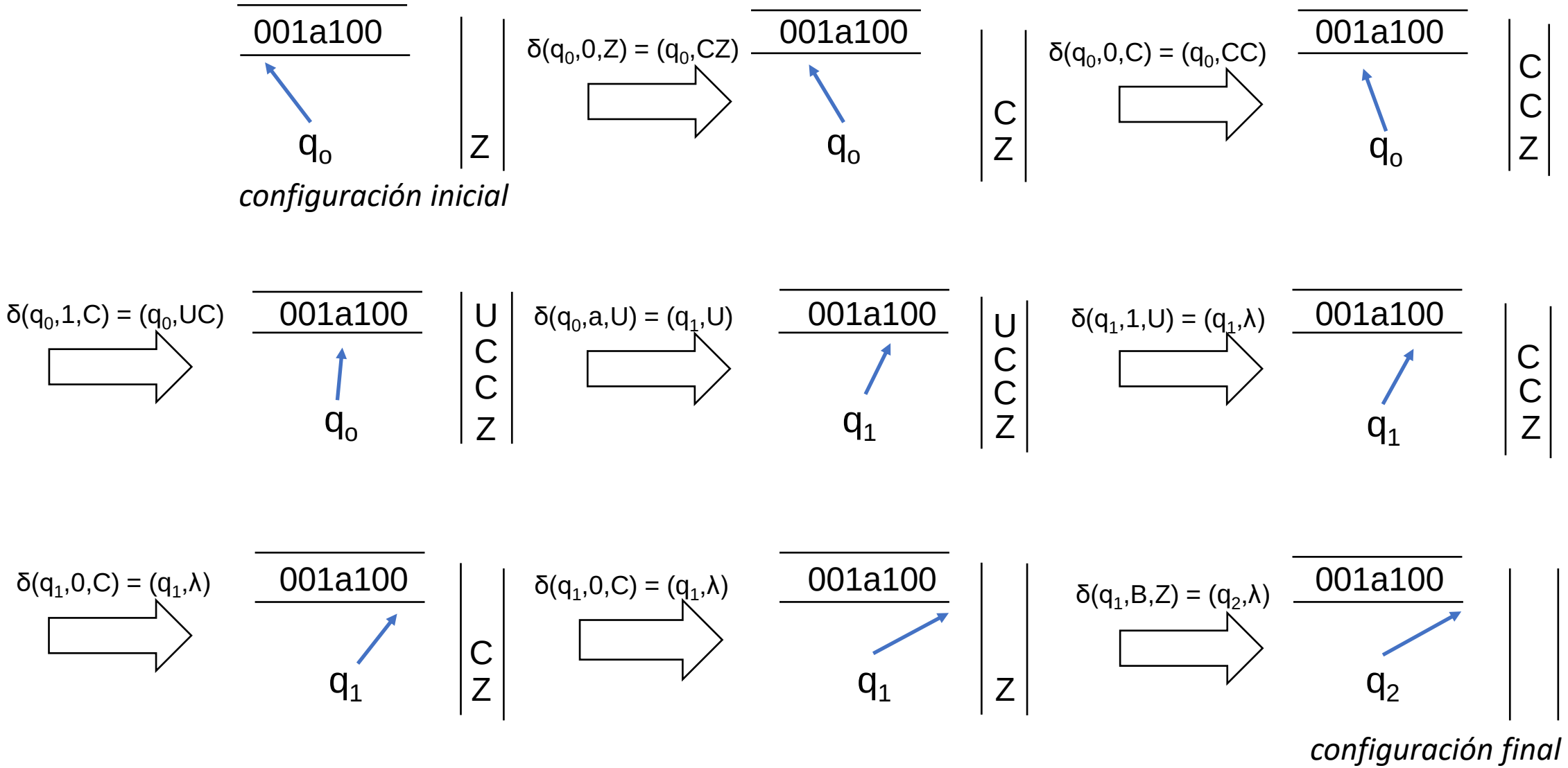
- Una primera cinta de entrada de sólo lectura.
- Una segunda cinta de lectura/escritura que se comporta como una pila.
- En un paso se pueden procesar las dos cintas.
- En la cinta de entrada siempre se va a la derecha.
- Cuando alcanza el símbolo B (blanco) en la cinta de entrada, el AP para y acepta sii la pila está vacía.

- Problemas típicos que resuelve un AP:
  - **Análisis sintáctico a nivel instrucción** de los compiladores.
  - **Evaluación de expresiones** en la ejecución de programas.



Ejemplo. Reconocimiento de palíndromos **waw^R,** tales que w tiene símbolos 0 y 1 y w^R es la inversa de w.

Supongamos la entrada **001a100**:



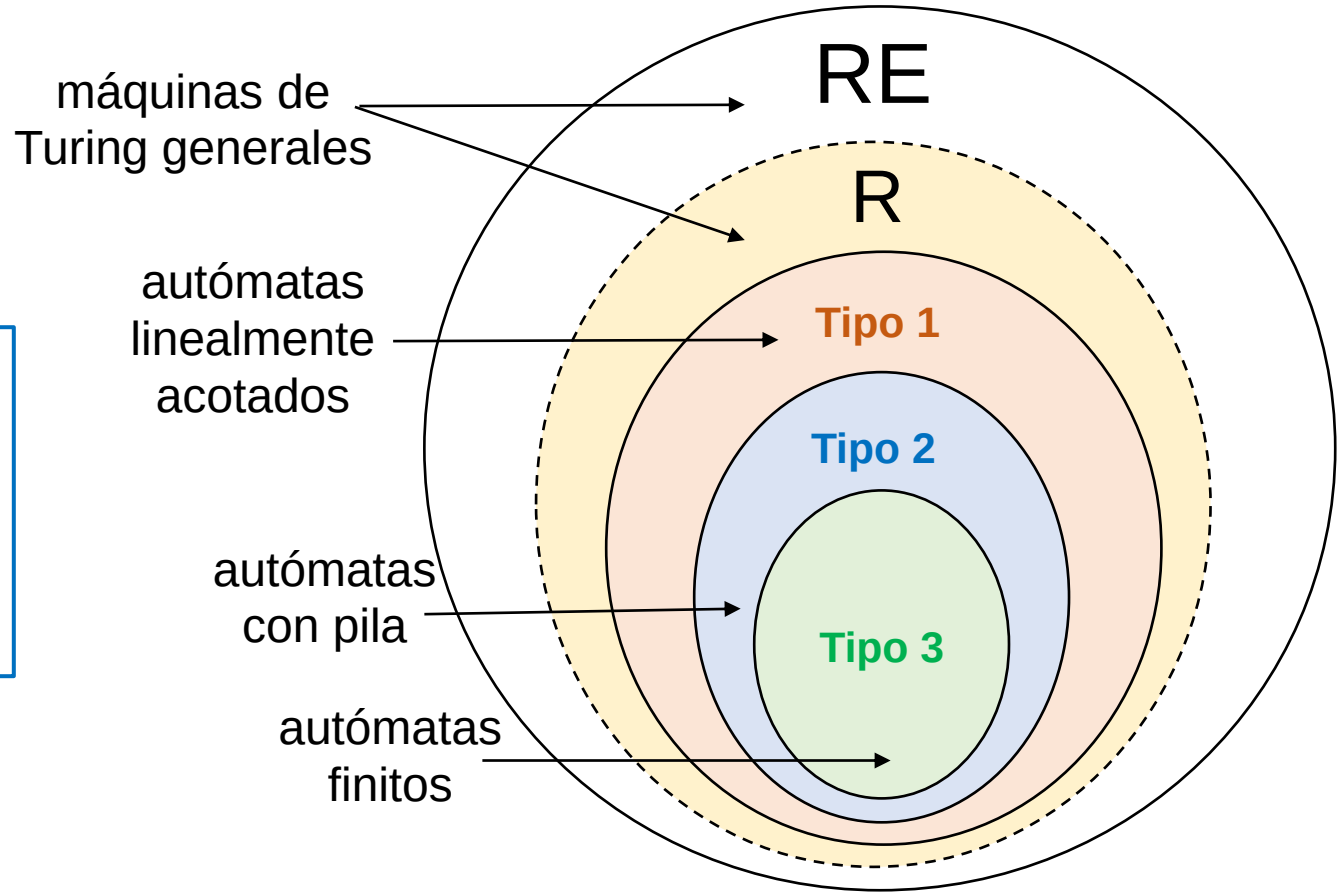
Algunas características de los AP

- Los AP **siempre paran.**
- Los lenguajes que aceptan se llaman **libres de contexto** o de **tipo 2.**
- A diferencia de las MT generales, **siempre se puede decidir si un AP:**

  Acepta una cadena w.
  Acepta el lenguaje ∅.
  Etc.

Jerarquía de lenguajes de Chomsky:

De tipo 0 (**recursivamente enumerables**)
De tipo 1 (**sensibles al contexto**)
De tipo 2 (**libres de contexto**)
De tipo 3 (**regulares**)



Enumeración de los lenguajes recursivamente enumerables

- **Todo lenguaje L de RE se puede enumerar:**

  Sea M_1 una MT que acepta L. Vamos a construir una MT M_2 que genera L:

  1. Hacer n := 1.
  2. Generar todas las cadenas de longitud a lo sumo n en el orden canónico.
  3. Por cada cadena generada ejecutar a lo sumo n pasos de la MT M_1. Si M_1 acepta, imprimir la cadena.
  4. Hacer n := n + 1 y volver al paso 2.

  ¿Las cadenas quedan en orden canónico? ¿Se pueden repetir? ¿Se puede evitar que se repitan? (ejercicio)

- **Todo lenguaje L de R se puede enumerar en el orden canónico:**

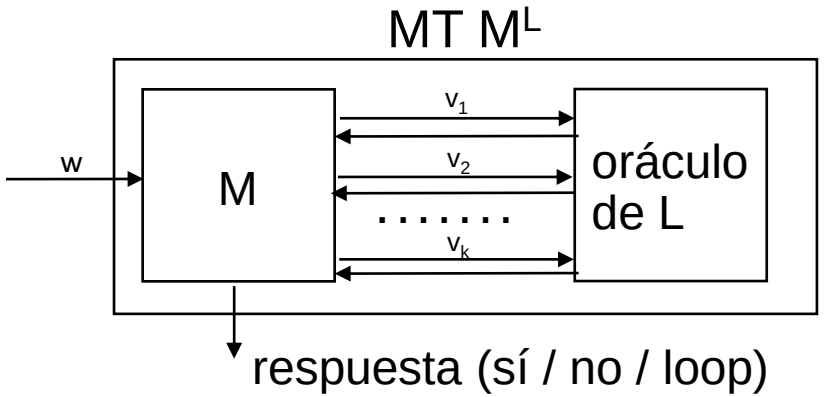
  Sea M_1 una MT que decide L. Vamos a construir una MT M_2 que genera L en el orden canónico:

  1. Generar la primera cadena en el orden canónico.
  2. Ejecutar M_1 sobre la cadena generada. Si acepta, imprimirla.
  3. Generar la siguiente cadena en el orden canónico y volver al paso 2.

- **Dada una MT que enumera un lenguaje, se puede construir otra MT que lo acepta** (ejercicio)

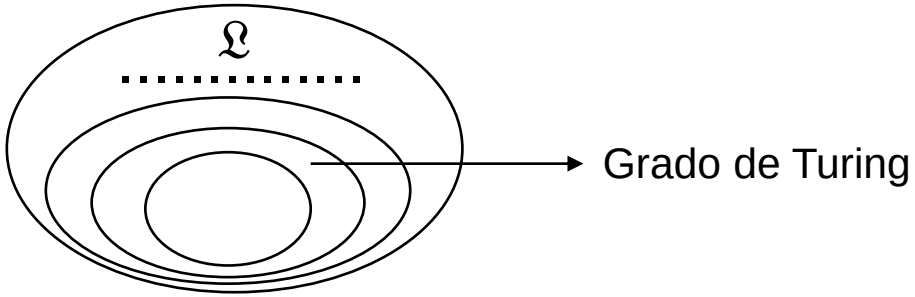
Turing-reducciones

- Las **Turing-reducciones** generalizan las reducciones descriptas antes (conocidas como m-reducciones).
- Se definen en términos de **MT con oráculo** (Turing, 1939):



- La MT M cuenta con un oráculo del lenguaje L, dispositivo capaz de responder en un paso si una cadena v pertenece o no pertenece a L.
- La MT y el oráculo en conjunto se denotan con $\mathbf{M^L}$.
- M puede invocar al oráculo cero o más veces.

- Se define que existe una Turing-reducción de L_1 a L_2 **($L_1 \leq_T L_2$)** sii existe una MT con oráculo de L_2 que decide L_1.
- Por ejemplo, se prueba que $\mathbf{L_\varnothing \leq_T L_U}$ y que $\mathbf{L_U \leq_T L_\varnothing}$. Esto significa que $\mathbf{L_\varnothing \in R}$ **sii** $\mathbf{L_U \in R}$. En este marco, se dice que $\mathbf{L_\varnothing}$ **y** $\mathbf{L_U}$ son **recursivamente equivalentes**.
- La propiedad de ser recursivamente equivalente determina una jerarquía de la computabilidad distinta de la que describimos antes. Se define en términos de **grados de Turing**.



Clase práctica 4

Ejemplo de reducción

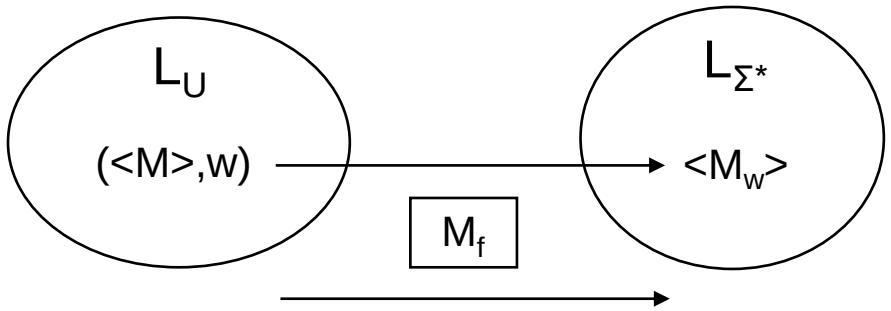
Sea el problema: dada una MT M, ¿acaso M acepta todas las cadenas de Σ*?
El lenguaje que representa el problema, sabemos, es: $L_{\Sigma^*} = \{<M> \mid L(M) = \Sigma^*\}$. Probaremos con una reducción que $\mathbf{L_{\Sigma^*} \notin R}$.
Ejercicio: Intuitivamente, ¿puede ser $L_{\Sigma^*} \in R$?. Más aún, ¿puede ser $L_{\Sigma^*} \in RE$?

Usaremos: **si $L_1 \leq L_2$ y $L_1 \notin R$, entonces $L_2 \notin R$.**
Así, hay que encontrar una reducción de la forma $L_1 \leq L_{\Sigma^*}$,
de modo tal que $L_1 \notin R$. Elegimos como L_1 el lenguaje $\mathbf{L_U}$.

Reducción:

Definimos: $f((<M>,w)) = <M_w>$, tal que M_w es una MT que:

a) Reemplaza su entrada por w.
b) Ejecuta M sobre w.
c) Acepta sii M acepta.



Computabilidad:

Existe una MT M_f que computa f: genera $<M_w>$, agregando al código <M> un fragmento inicial que borra su entrada y la reemplaza por w.

Correctitud:

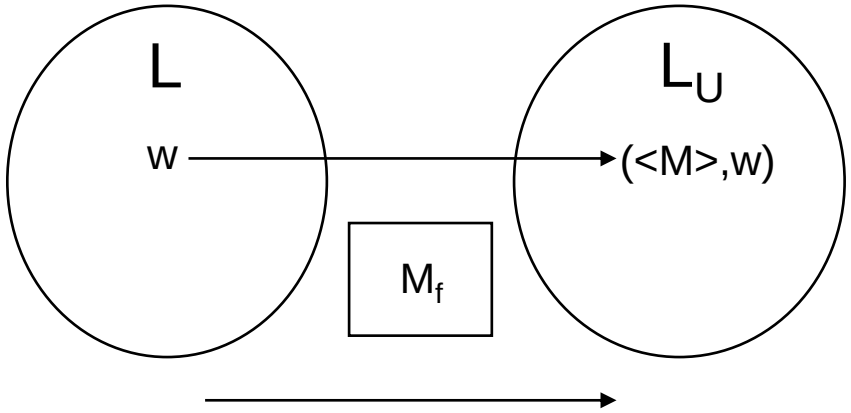
$(<M>, w) \in L_U \longrightarrow$ M acepta w $\longrightarrow$ M_w acepta todas sus entradas (¿por qué?) $\longrightarrow L(M_w) = \Sigma^* \longrightarrow <M_w> \in L_{\Sigma^*}$
$(<M>, w) \notin L_U \longrightarrow$ caso cadena válida**:** M rechaza w $\longrightarrow$ M_w rechaza todas sus entradas (¿por qué?) $\longrightarrow L(M_w) \neq \Sigma^* \longrightarrow <M_w> \notin L_{\Sigma^*}$
caso cadena no válida**:** M_w tampoco es una cadena válida $\longrightarrow L(M_w) \neq \Sigma^* \longrightarrow <M_w> \notin L_{\Sigma^*}$

No simetría de las reducciones

Las reducciones en general no cumplen la propiedad de simetría:

- Sea cualquier lenguaje L ∈ R. Y sea M una MT que decide L.
- Por un lado, no se cumple $\mathbf{L_U \leq L}$ (¿por qué?)
- Pero por otro lado, sí se cumple $\mathbf{L \leq L_U}$:

Reducción:



Computabilidad: existe una MT M_f que, dada una cadena w, le concatena a la izquierda el código <M>

Correctitud: w ∈ L sii M acepta w sii (<M>, w) ∈ $\mathbf{L_U}$

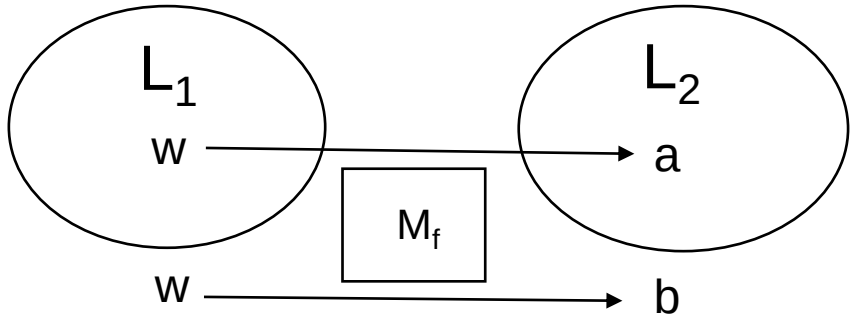
Simetría de las reducciones dentro de la clase R

- En particular, en R se cumple que las reducciones son simétricas (sin considerar los lenguajes especiales Σ^* y $\varnothing$).
- En otras palabras, en el marco de la computabilidad **todos los lenguajes de R tienen la misma dificultad**.
- La prueba es la siguiente:

Sean L_1 y L_2 distintos de Σ^* y $\varnothing$.
Sean $a \in L_2$ y $b \notin L_2$.
Y sean M_1 y M_2 dos MT que deciden L_1 y L_2, respectivamente.
Veamos que $\mathbf{L_1 \leq L_2}$:

Reducción:

$f(w) = a$ si $w \in L_1$
$f(w) = b$ si $w \notin L_1$



Computabilidad:

Dada w, la MT M_f que computa f ejecuta M_1 sobre w,
si acepta w genera *a* y si rechaza w genera b.

Correctitud:

$w \in L_1$ sii $f(w) \in L_2$