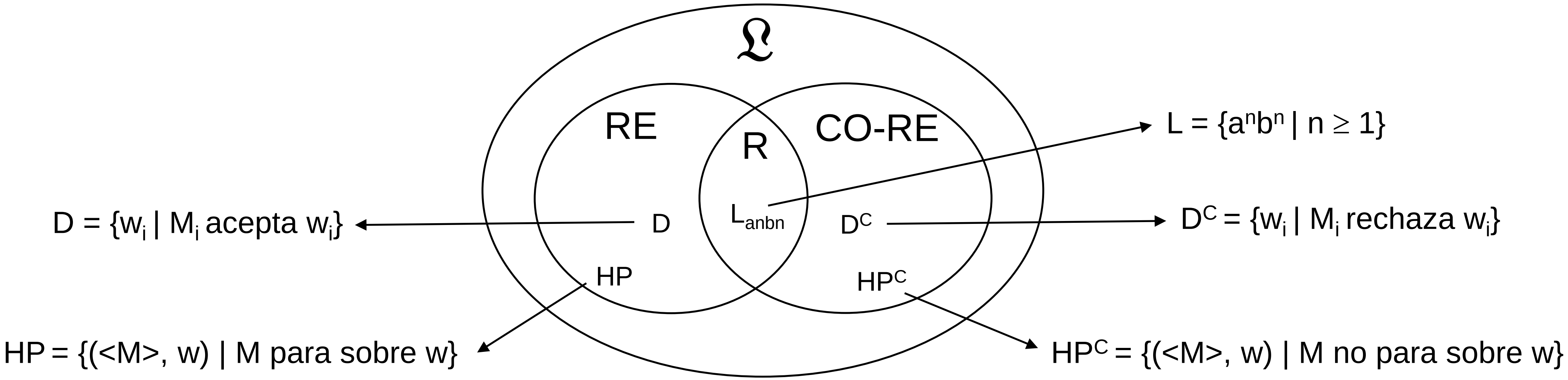
Clase teórica 4

Reducciones de Problemas
Misceláneas de Computabilidad

Repaso

- Poblamos la jerarquía de la computabilidad con algunos primeros lenguajes:



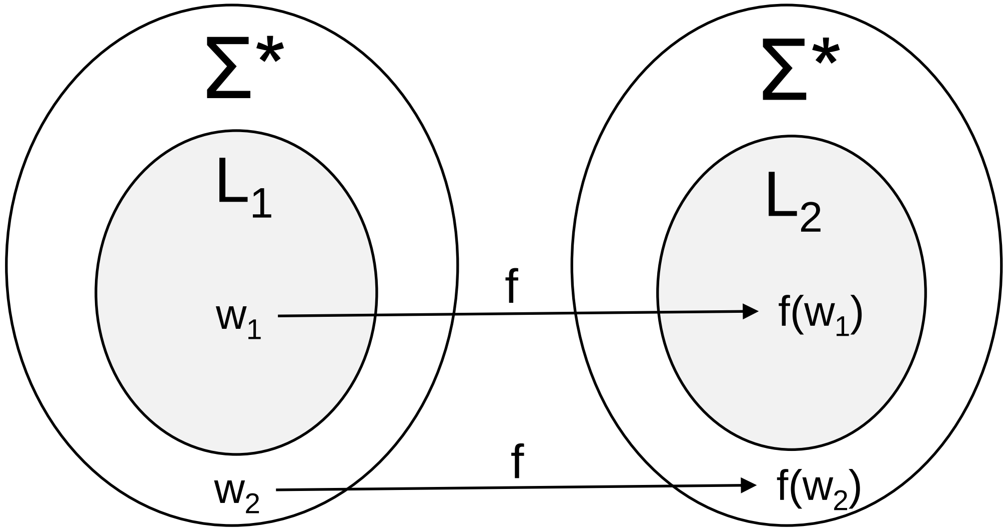
- Para probar **pertenencia a R y RE** hemos **construido MT.**
- Para probar **no pertenencia a R y RE** hemos utilizado la técnica de **diagonalización.**
- Una técnica más sencilla para probar **no pertenencia a R y RE** es la **reducción**.

Reducciones de lenguajes

- Dados dos lenguajes L_1 y L_2, supongamos que existe una MT M_f que computa una función $f : \Sigma^* \longrightarrow \Sigma^*$ de la siguiente forma:

a partir de todo $w \in L_1$, la MT M_f genera $f(w) \in L_2$

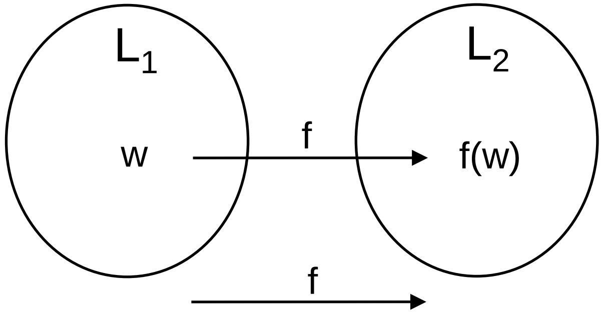
a partir de todo $w \notin L_1$, la MT M_f genera $f(w) \notin L_2$



Se define que la función f es una **reducción** de L_1 a L_2.

Se anota $\mathbf{L_1 \leq L_2}$, y se dice que la función f es **total computable** (se computa sobre todas las cadenas).

Utilidad de las reducciones



Reducción de L_1 a L_2

Para todo w, $w \in L_1$ sii $f(w) \in L_2$

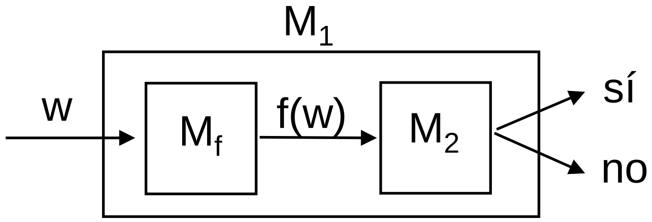
TEOREMA

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 ($L_1 \notin R \longrightarrow L_2 \notin R$)

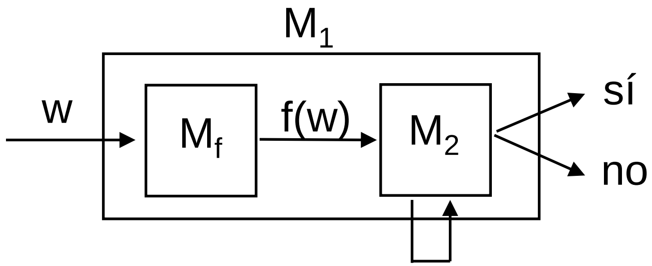


Caso 2

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 ($L_1 \notin RE \longrightarrow L_2 \notin RE$)



En ambos casos, $w \in L_1$ sii $f(w) \in L_2$

Por eso M_1 responde lo que responde M_2

Es decir, si $L_1 \leq L_2$:

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

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

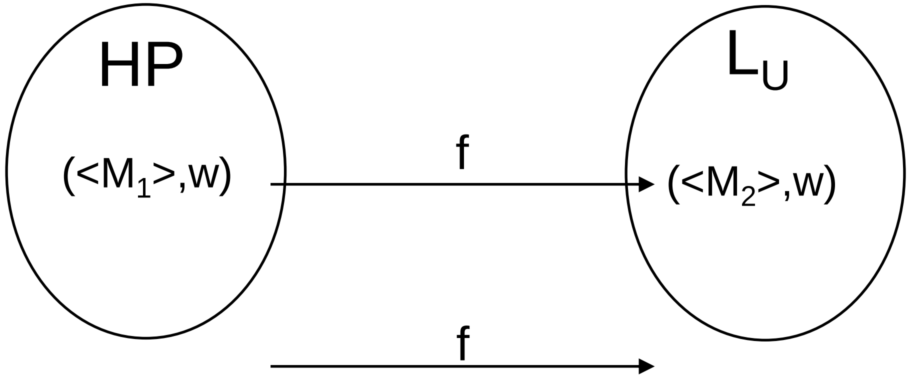
L_2 es **tan o más difícil** que L_1, **resolviendo L_2 se resuelve L_1.**

De esta manera, las reducciones permiten encontrar lenguajes **dentro y fuera de R y RE.**

- **Ejemplo 1.**

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

Vamos a probar HP ≤ L_U:



De acuerdo al teorema.

si $L_1 \leq L_2$, entonces $L_1 \notin R \longrightarrow L_2 \notin R$.

Por lo tanto, como HP $\notin$ R,

también probamos que $L_U \notin R$.

Definición de la reducción
$f((<M_1>, w)) = (<M_2>, w)$, con M_2 como M_1, salvo que **los estados q_R de M_1 se cambian en M_2 por estados q_A.**

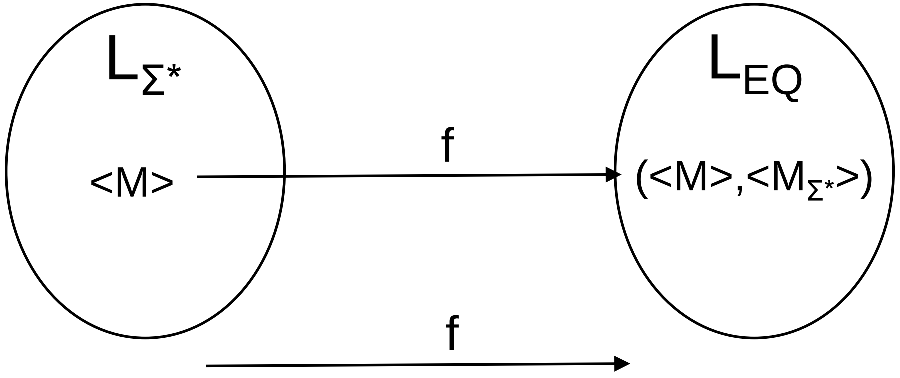
Computabilidad
Existe una MT M_f que computa f: copia $(<M_1>, w)$ pero cambiando los estados q_R de M_1 por estados q_A en M_2.

Correctitud
$(<M_1>, w) \in HP \longrightarrow M_1$ para sobre w $\longrightarrow M_2$ acepta w $\longrightarrow (<M_2>, w) \in L_U$
$(<M_1>, w) \notin HP \longrightarrow$ <u>caso de cadena válida</u>: M_1 no para sobre w $\longrightarrow M_2$ no para sobre w $\longrightarrow (<M_2>, w) \notin L_U$
<u>caso de cadena inválida</u>: $(<M_2>, w)$ también es una cadena inválida $\longrightarrow (<M_2>, w) \notin L_U$

- **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 $L_{\Sigma^*} \leq L_{EQ}$:



De acuerdo al teorema,

si $L_1 \leq L_2$, entonces $L_1 \notin RE \longrightarrow L_2 \notin RE$.

Por lo tanto, como $L_{\Sigma^*} \notin RE$,

también probamos que $L_{EQ} \notin RE$.

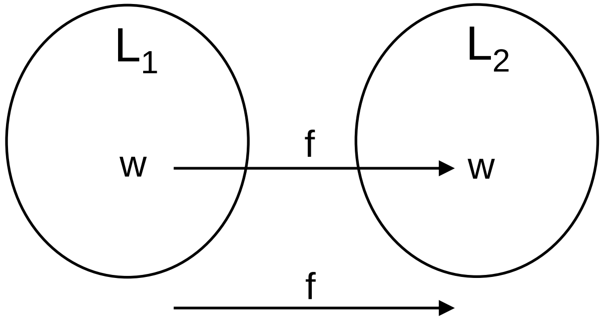
Definición de la reducción
$f(<M>) = (<M>, <M_{\Sigma^*}>)$, tal que $L(M_{\Sigma^*}) = \Sigma^*$.

Computabilidad
Existe una MT M_f que computa f: copia <M> y le concatena <M_{Σ^*}>.

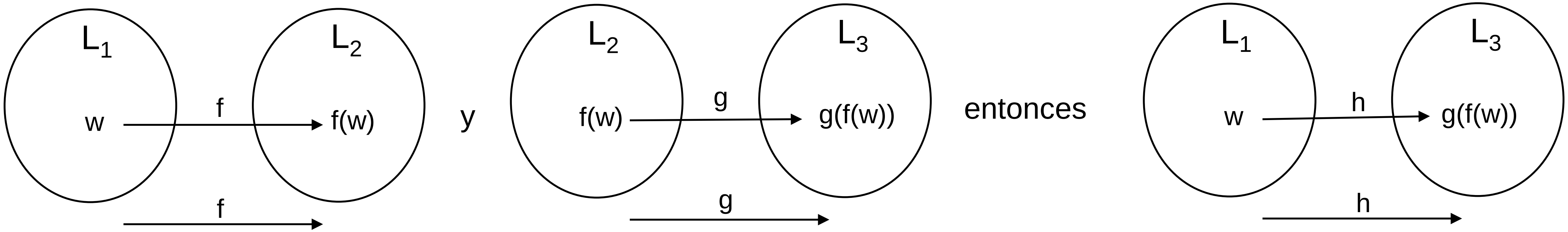
Correctitud
$<M> \in L_{\Sigma^*} \longrightarrow L(M) = \Sigma^* \longrightarrow L(M) = L(M_{\Sigma^*}) \longrightarrow (<M>, <M_{\Sigma^*}>) \in L_{EQ}$
$<M> \notin L_{\Sigma^*} \longrightarrow$ <u>caso de cadena válida</u>: $L(M) \neq \Sigma^* \longrightarrow L(M) \neq L(M_{\Sigma^*}) \longrightarrow (<M>, <M_{\Sigma^*}>) \notin L_{EQ}$
<u>caso de cadena inválida</u>: $(<M>, <M_{\Sigma^*}>)$ también es una cadena inválida $\longrightarrow (<M>, <M_{\Sigma^*}>) \notin L_{EQ}$

Algunas propiedades de las reducciones

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

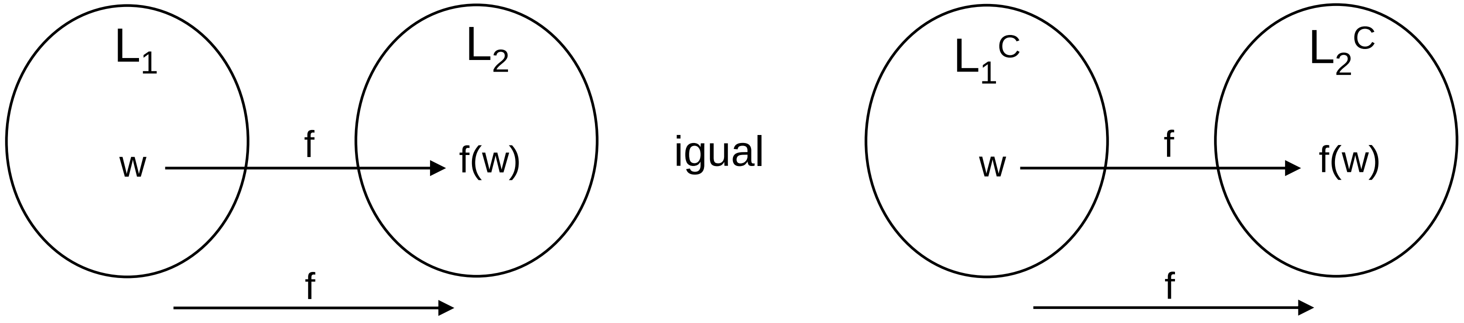


- **Transitividad.** Si $L_1 \leq L_2$ y $L_2 \leq L_3$, entonces $L_1 \leq L_3$.



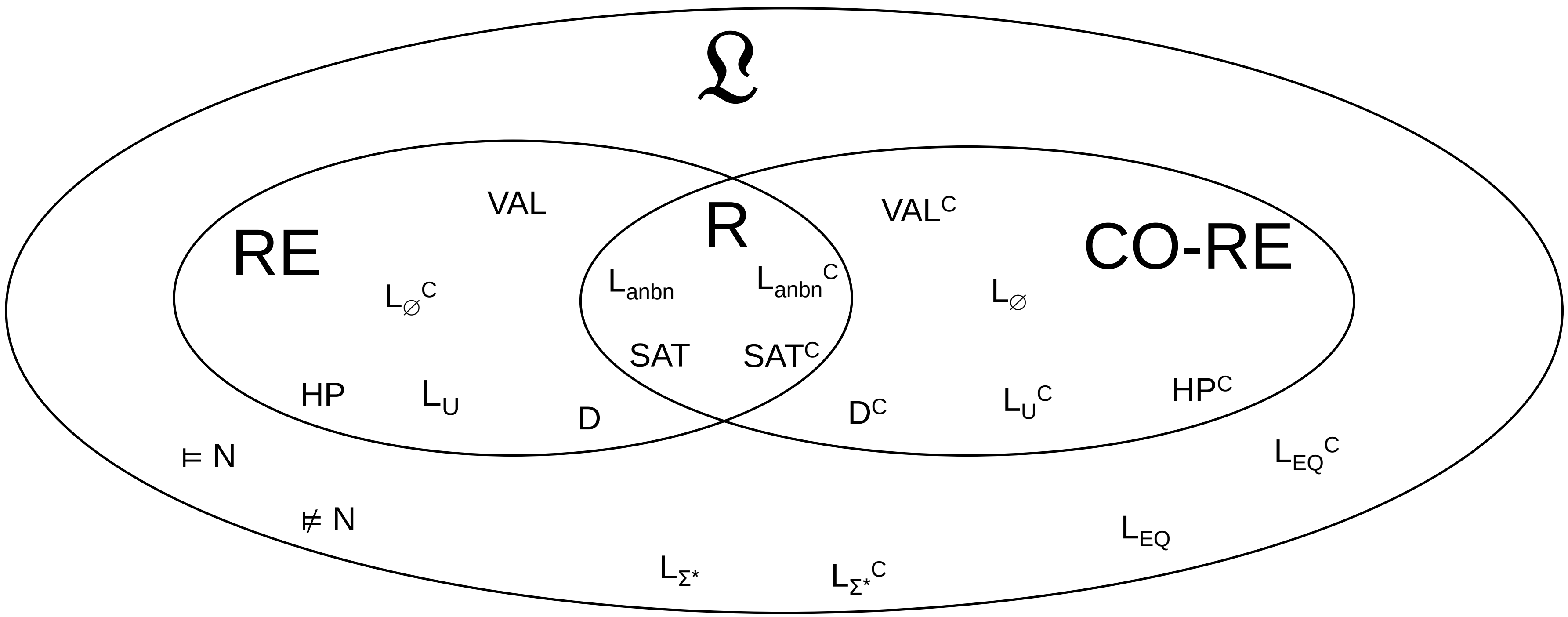
Se componen las reducciones f y g y se obtiene la reducción h.

- **Otra propiedad:** $L_1 \leq L_2$ sii $L_1^C \leq L_2^C$. Es la misma función de reducción.



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

Ultima mirada a la jerarquía de la computabilidad



SAT: fórmulas booleanas satisfactibles.

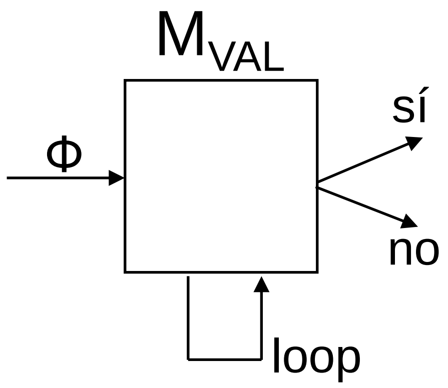
VAL: fórmulas válidas (teoremas) de la lógica de predicados.

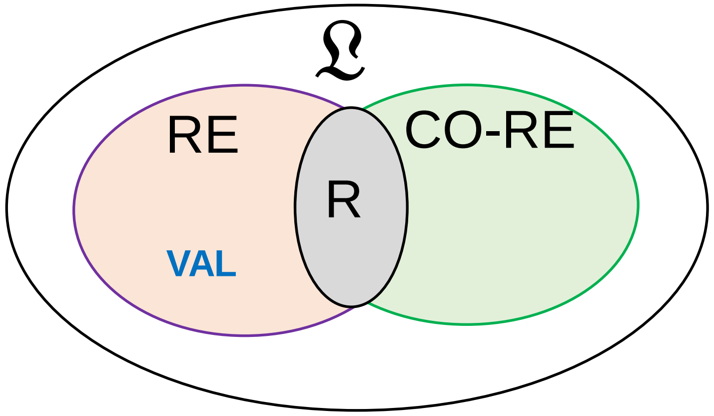
$\models$ N: enunciados verdaderos de la aritmética.

Indecibilidad de la lógica de predicados revisitada (Turing, 1936)

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



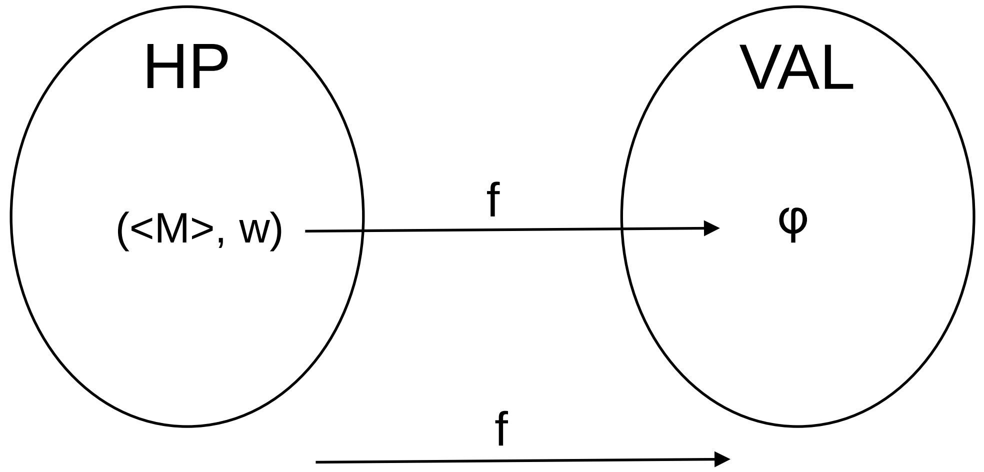


Ejemplo: $\Phi = \neg P \rightarrow (P \rightarrow Q)$

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

Indecibilidad de la lógica de predicados revisitada (Turing, 1936)

- Reducción de HP a VAL:



Para todo par (<M>, w),

si **M para desde w**, f((<M>, w)) = **φ es una fórmula válida (teorema) de la lógica de predicados**

si **M no para desde w**, f((<M>, w)) = **φ no es una fórmula válida (teorema) de la lógica de predicados**

- **Si la lógica de predicados fuera decidible, sería decidible la detención de las máquinas de Turing.**

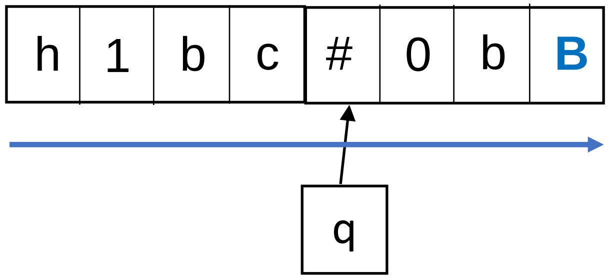
Anexo de la clase teórica 4

Reducciones de Problemas
Misceláneas de Computabilidad

Máquinas de Turing restringidas

AUTÓMATAS FINITOS (AF) (1943: modelo neuronal de McCulloch-Pitts)

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



- Cuando se alcanza el símbolo B (blanco) el AF para (acepta sii el estado alcanzado es 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. $\delta(q_0, 0) = q_2$

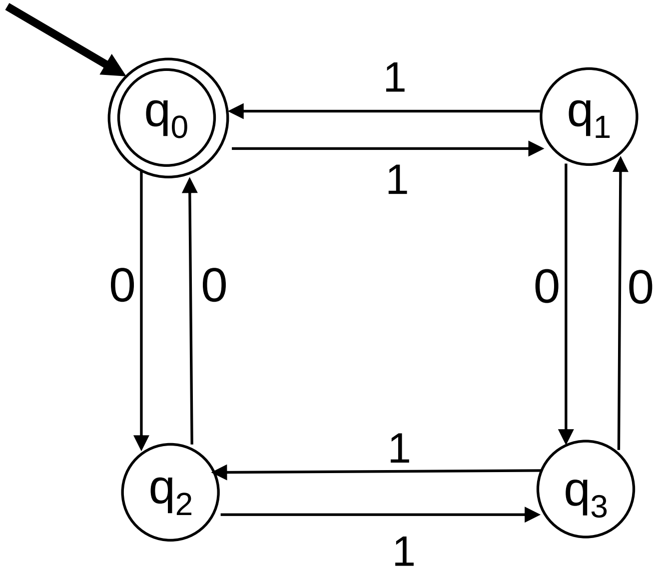


Diagrama de transición de estados

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.**

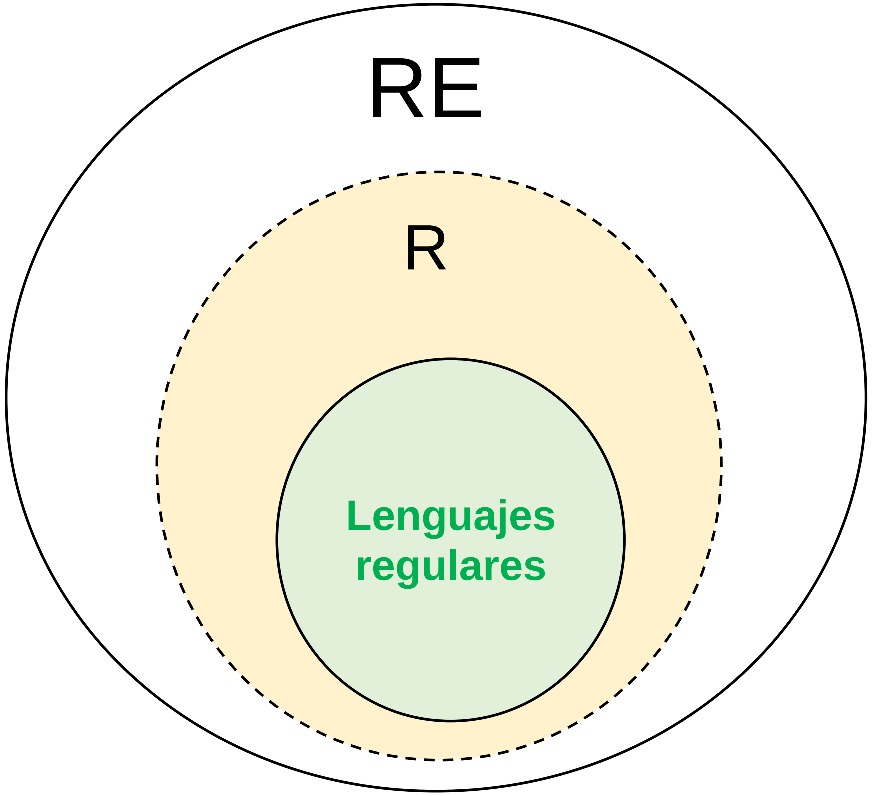
- Aceptan un tipo limitado de cadenas **(no tienen memoria).**

  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 se llaman **regulares** o de **tipo 3**.

- A diferencia de las MT generales, **pueden decidir**:

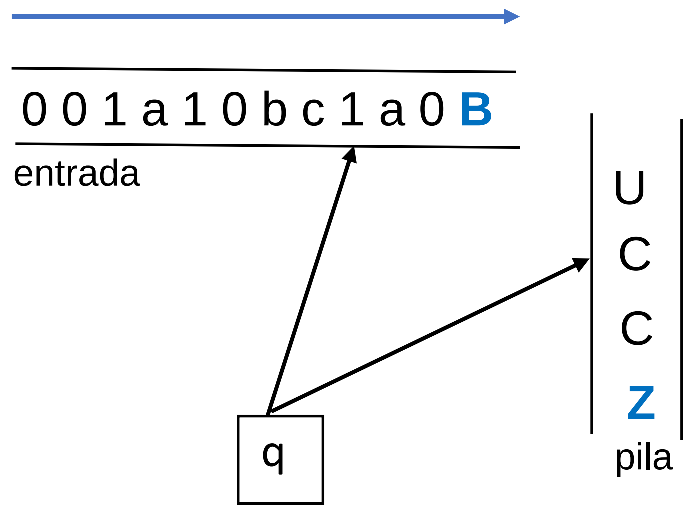
  ¿$w \in L(M)$?
  ¿$L(M) = \emptyset$?
  ¿$L(M) = \Sigma^*$?
  ¿$L(M) = L(M')$?



AUTÓMATAS CON PILA (AP)

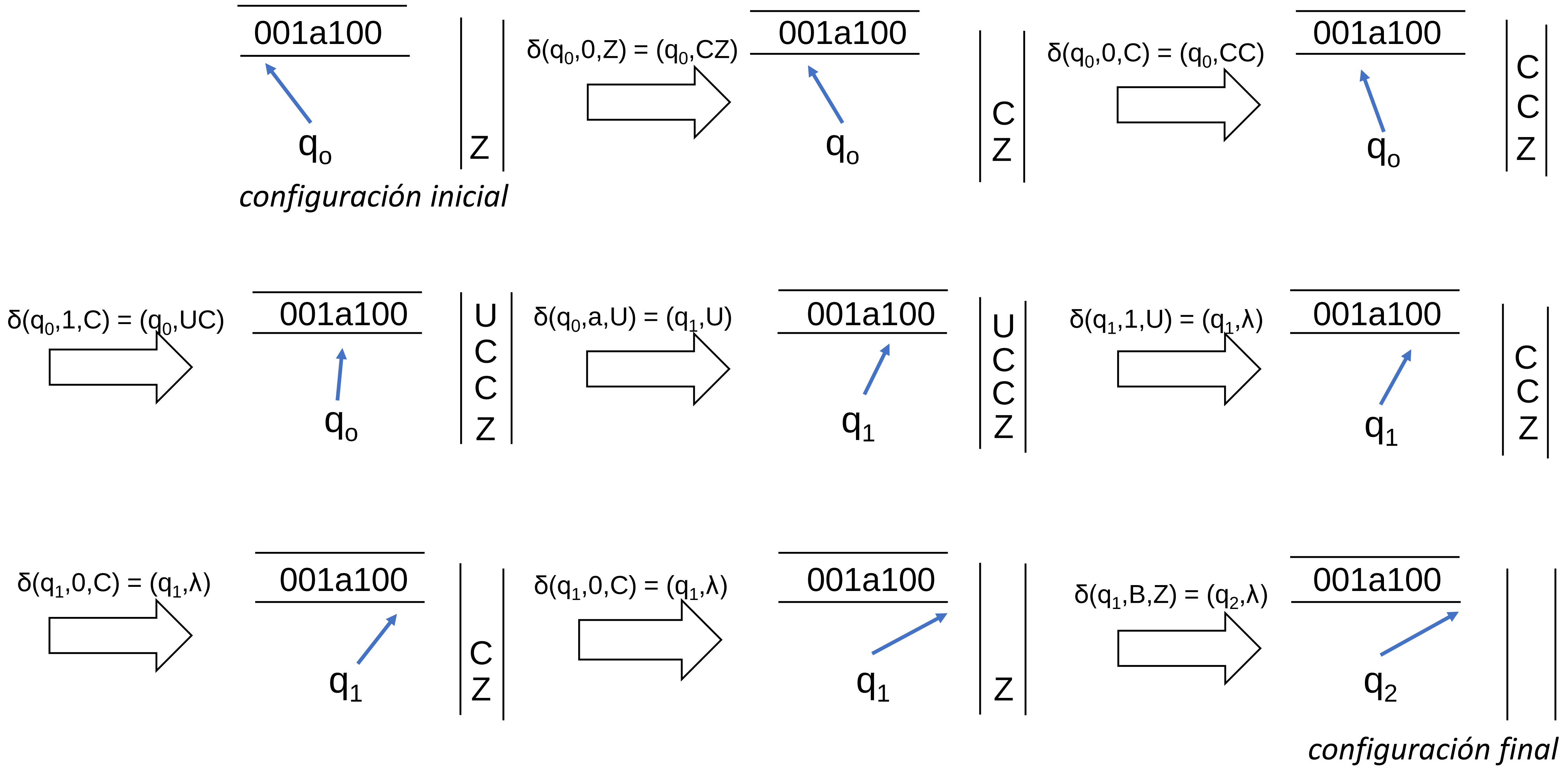
- Una cinta de input de sólo lectura.
- Una cinta de lectura/escritura que se comporta como una pila.
- En un paso se pueden procesar las dos cintas.
- En la cinta de entrada siempre se va a la derecha.
- Cuando se alcanza el símbolo B (blanco) en la cinta de entrada, el AP para (acepta sii la pila está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

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



<u>Ejemplo</u>. Reconocimiento de cadenas **waw^C,** tales que w tiene 0 y 1 y w^C es la inversa de w.

Supongamos la entrada **001a100**:



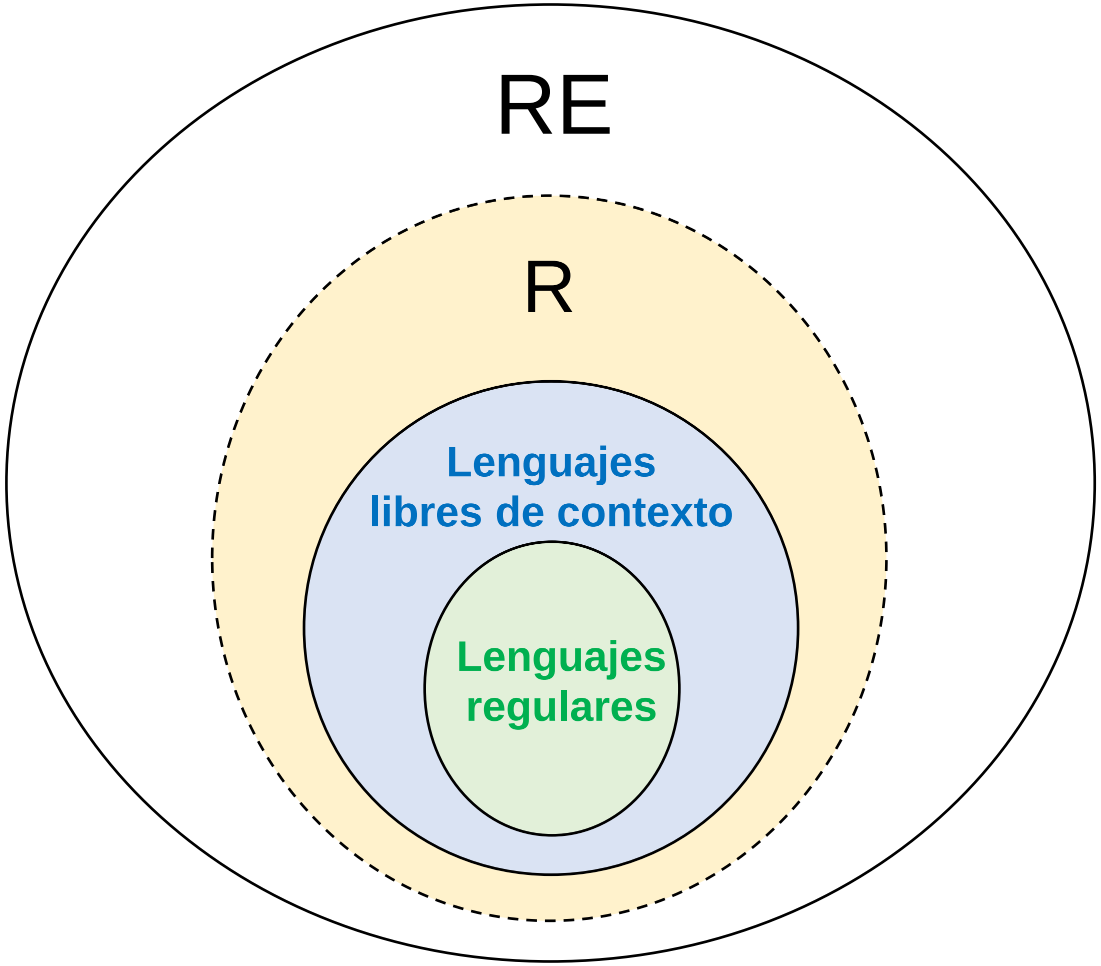
Algunas características de los AF

- Los AP **siempre paran.**

- Los lenguajes que aceptan se llaman **libres de contexto** o de **tipo 2.**

- A diferencia de las MT generales, **pueden decidir**:

  ¿$w \in L(M)$?
  ¿$L(M) = \varnothing$?



- **Jerarquía de Chomsky:**

  Lenguajes de tipo 3 (**regulares**)
  Lenguajes de tipo 2 (**libres de contexto**)
  Lenguajes de tipo 1 (**sensibles al contexto**)
  Lenguajes de tipo 0 (**recursivamente numerables**)

Enumeración de los lenguajes recursivamente numerables

- **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**)**

Clase práctica 4

Reducciones de Problemas

Misceláneas de Computabilidad

Ejemplo 1 de reducción.

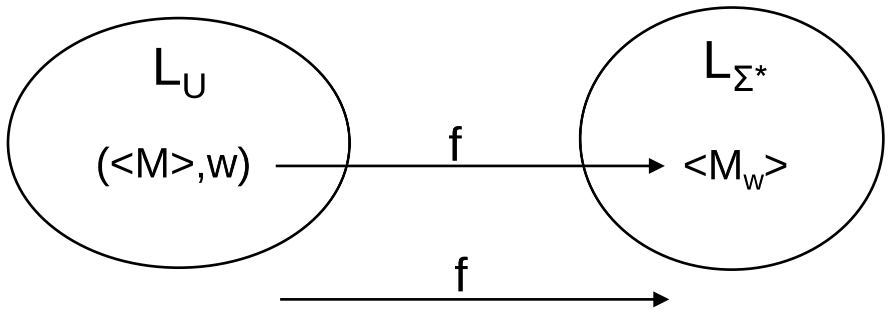
Sea el problema: dada una MT M, ¿acaso M acepta todas las cadenas de Σ^*?
El lenguaje que representa el problema es: $L_{\Sigma^*} = \{<M> \mid L(M) = \Sigma^*\}$. Probaremos con una reducción que $\mathbf{L_{\Sigma^*} \notin R}$.
Ejercicio: Intuitivamente, ¿puede ser $L_{\Sigma^*} \in R$?. Más aún,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

Definición de la reducción:

Se define: $f((<M>,w)) = <M_w>$, 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

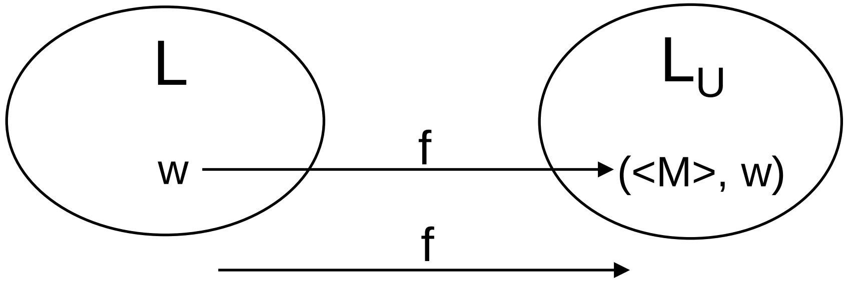
Existe una MT M_f que computa f: genera $<M_w>$, agregando al código $<M>$ un fragmento inicial que borra su entrada y la reemplaza por w.

Correctitud

$(<M>, w) \in L_U \longrightarrow$ M acepta w $\longrightarrow M_w$ acepta todas sus entradas $\longrightarrow L(M_w) = \Sigma^* \longrightarrow <M_w> \in L_{\Sigma^*}$
$(<M>, w) \notin L_U \longrightarrow$ <u>caso de cadena válida</u>**:** M rechaza w $\longrightarrow M_w$ rechaza todas sus entradas $\longrightarrow L(M_w) \neq \Sigma^* \longrightarrow <M_w> \notin L_{\Sigma^*}$
<u>caso de cadena inválida</u>**:** M_w también es una cadena inválida $\longrightarrow L(M_w) \neq \Sigma^* \longrightarrow <M_w> \notin L_{\Sigma^*}$

<u>Ejemplo 2 de reducción</u>.

L es algún lenguaje de RE.
L_U = {(<M>, w) | M acepta w}.
Vamos a probar que **hay una reducción de L a L_U**:



Definición de la reducción

Sea M una MT que acepta L (¿por qué existe M?).
Hacemos, para todo w: f(w) = (<M>, w)

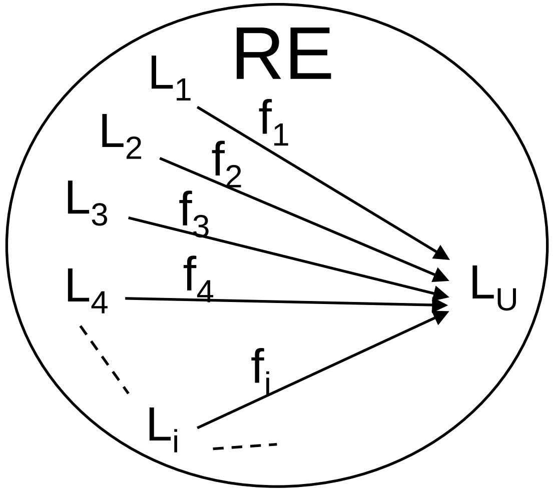
Computabilidad

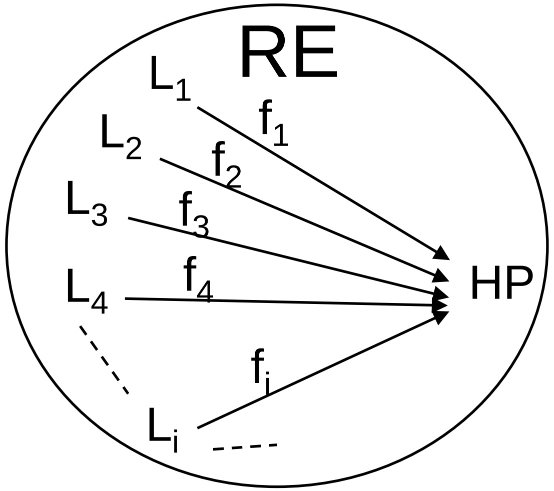
Existe una MT M_f que computa f: dado w, devuelve <M> concatenado con w.

Correctitud

Claramente, w ∈ L sii (<M>, w) ∈ L_U (¿por qué?).

Hemos probado en definitiva que todo lenguaje de RE se puede reducir a L_U. Lo mismo ocurre con HP.





Los lenguajes L_U y HP
son **de los más difíciles** de la clase RE.

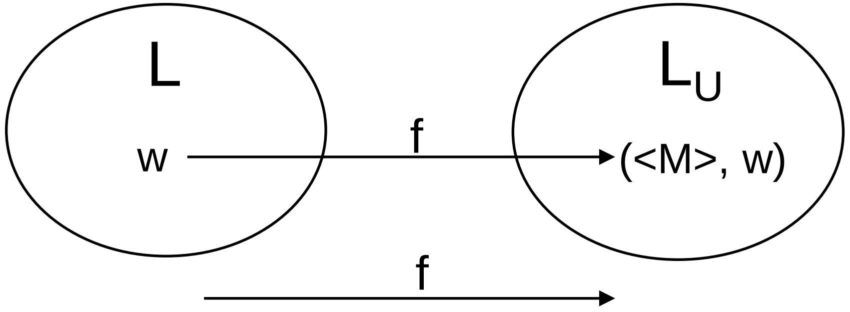
Todos los lenguajes de RE se reducen a ellos.

Si L_U o HP fueran recursivos, entonces se cumpliría la igualdad R = RE.

<u>Ejemplo 3. No simetría de las reducciones</u>.

Las reducciones no cumplen la propiedad de simetría.

- Sea cualquier $L \in R$. Sea M una MT que decide.
- No puede ser $\mathbf{L_U} \leq \mathbf{L}$ (¿por qué?).
- Pero se cumple $\mathbf{L} \leq \mathbf{L_U}$:



(Caso particular de lo visto en el slide anterior)

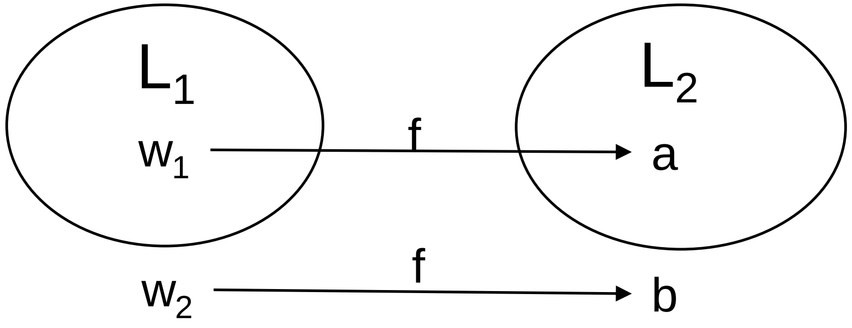
Ejemplo 4. Propiedad de las reducciones en la clase R.

A diferencia de lo que sucede en la clase RE, en la clase R se cumple, sin considerar los lenguajes especiales Σ^* y $\emptyset$, que cualquier lenguaje L_1 se puede reducir a cualquier lenguaje L_2.
En otras palabras, todos los lenguajes de R tienen la misma dificultad.
La prueba es la siguiente:

Sean L_1 y L_2 distintos de Σ^* y $\emptyset$.
Sean $a \in L_2$ y $b \notin L_2$.
Sean M_1 y M_2 tales que deciden L_1 y L_2.

Definición de la reducción

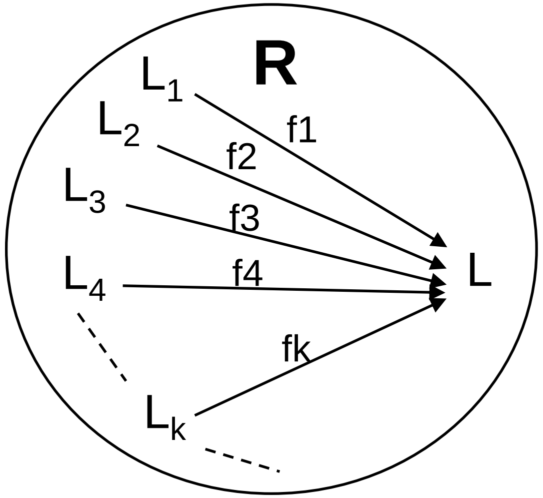
$f(w) = a$ si $w \in L_1$
$f(w) = b$ si $w \notin L_1$



Computabilidad

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

Todos los lenguajes de R tienen igual dificultad.



Correctitud

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