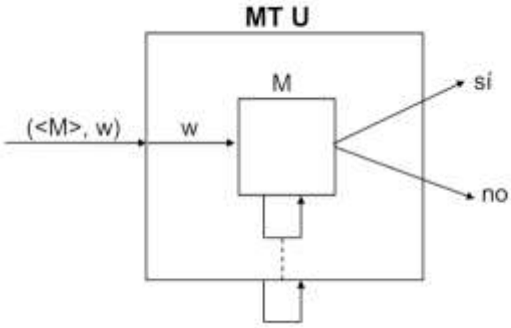
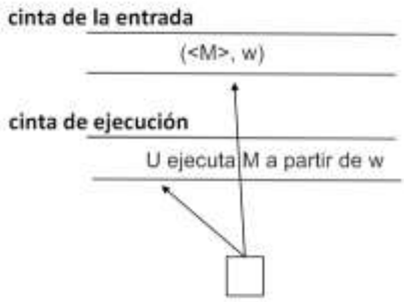
Practica 2 – Clases 3 – 4

MT Universal

Es una máquina de Turing capaz de ejecutar otra





- La MT U recibe como entrada una MT M (codificada mediante una cadena) y una cadena w, y ejecuta M a partir de w. Puede tener una o más cintas de ejecución.

Se usan pruebas constructivas para probar pertenencia a R o a RE es básicamente construir una MT que acepte el lenguaje.

Para probar no pertenencia a R o a RE se tienen pruebas no constructivas, para esto hace falta encontrar un primer lenguaje L en RE – R, para probar $R \subset RE$ y un primer lenguaje L^C en CO-RE – R, para probar $RE \subset \mathfrak{L}$.

Para numerar las cadenas se utiliza el orden canónico:

1) De menor a mayor longitud.

2) Con longitudes iguales desempata el orden alfanumérico (números, letras y símbolos especiales).

Se tiene una MT M que obtiene una MT Mi según el orden canónico: lo que hace básicamente es generar las cadenas en orden canónico y devuelve la cadena siempre y cuando sea una codigo valido de MT y el contador que mantiene (n) sea igual a i. (las que no son MT validas las descarta, ni si quiera las enumera).

La tabla T representa el comportamiento de todas las MT con respecto a todas las cadenas:

orden canónico

orden canónico

T	w_0	w_1	w_2	w_3	w_4	......
M_0	1	0	1	1	1	......
M_1	1	0	0	1	0	......
M_2	0	0	1	0	1	......
M_3	0	1	1	1	1	......
M_4	0	1	1	1	0	......
......	......	......	......	......	......	......

T(Mi , wj) = 1 significa que Mi acepta wj , y T(Mi , wj) = 0 significa que Mi rechaza wj

Cada fila de T representa un lenguaje de RE. Por lo tanto, el conjunto de las filas de T representa el conjunto RE.

L(M0) = {w0, w2, w3, w4, ...}

L(M1) = {w0, w3, ...}

RE → conjuntos de lenguajes aceptados por maquinas, con lo cual RE es el conjunto de los lenguajes L(M0), L(M1), etc...

La diagonal de T representa otro lenguaje: D = {wi | Mi acepta wi }.

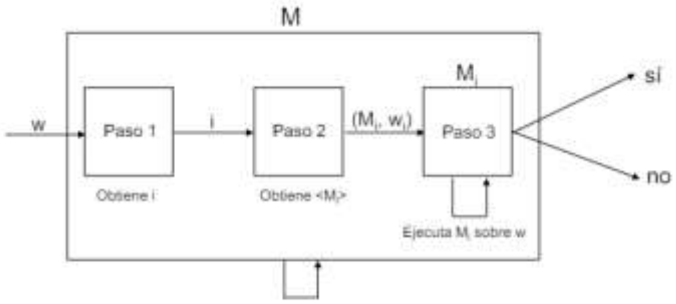
T	w_0	w_1	w_2	w_3	w_4	.....
M_0	**1**	0	1	1	1	.....
M_1	1	**0**	0	1	0	.....
M_2	0	0	**1**	0	1	.....
M_3	0	1	1	**1**	1	.....
M_4	0	1	1	1	**0**	.....
.....	.....	.....	.....	....	.....	**.....**

Y la diagonal de T con los 1 y 0 invertidos este otro: D^C = {wi | Mi rechaza wi }.

D está en RE

Vamos a construir una MT M que acepta el lenguaje D. Dada una entrada w, M hace:

1. Encuentra i tal que $w = w_i$ en el orden canónico (genera cadenas en orden canónico hasta encontrar w).
2. Encuentra el código <M_i> de la MT M_i (genera códigos de MT en orden canónico hasta llegar al i-ésimo).
3. Ejecuta M_i sobre w_i y acepta sii M_i acepta.



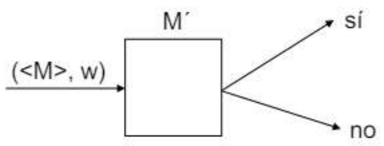
D^C no está en RE y por lo tanto --> D no está en R

- No esta en RE porque todas las filas difieren de d^c en algún elemento (fila 0 – 1er, fila 1 – 2do, fila 2 – 3ro, etc), por lo tanto todos los lenguajes de RE difieren del lenguaje D^C
- En este caso no sirve construir una MT.

Asi se encontraron 2 lenguajes fuera de R. A partir de estos primeros lenguajes, encontrados por diagonalización, se puede poblar la jerarquía de la computabilidad con una técnica más sencilla: la reducción

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} (está en RE – R). Es de los más difíciles de RE. Si HP estuviera en R, todo lenguaje de RE estaría en R (sería RE = R).



M' primero decide si M para a partir de w.
Si detecta que M no para, M´ rechaza.
Si detecta que M para, M´ ejecuta M desde w y responde como M.

Se puede probar por diagonalización (G. Cantor) que el conjunto R de los números reales es más grande que el conjunto N de los naturales.

Otra manera de probar RE $\subset \mathfrak{L}$.

No puede haber más MT, y así lenguajes recursivamente numerables, que $|\Sigma^*|$; la cantidad de lenguajes de es $|P(\Sigma^*)|$; y como $|\Sigma^*| < |P(\Sigma^*)|$, entonces RE $\subset \mathfrak{L}$. ($|RE| < |\mathfrak{L}|$)

Recordar

$\Sigma^* \rightarrow$ conjunto de todas las palabras formadas por símbolos de Σ (incluido λ)

$\mathfrak{L} \rightarrow$ el conjunto de todos los lenguajes definidos sobre el alfabeto Σ ($P(\Sigma^*)$)

Cómo burlar al halting problema

Supongamos que existe una MT MHP que decide HP. Llegaremos a una contradicción. Construimos una MT P de la siguiente manera. Dada una entrada <Q> la MT P hace:

1. Ejecuta MHP sobre (<Q>, <Q>).

2. Si MHP responde que sí, entonces P "se hace entrar en un loop".

3. Si MHP responde que no, entonces P para (responde indistintamente sí o no).

P "le lleva la contra" a MHP .

Veamos qué sucede cuando la entrada de P es su propio código <P> :

1. Si P para desde <P>, significa que MHP respondió no desde (<P>, <P>), es decir que P no para desde <P> .
2. Si P no para desde <P>, significa que MHP respondió sí desde (<P>, <P>), es decir que P para desde <P> .

Ambos casos se obtiene una contradicción. Por lo tanto, no puede existir P. Y como P se construyó a partir de MHP, entonces tampoco puede existir MHP . Es decir, HP no es recursivo.

La computabilidad de un lenguaje tiene más que ver con su definición, su contorno (representación gráfica), que con su tamaño. Ejemplo: HP $\subseteq \Sigma^*$, pero HP es más difícil que Σ^* (HP $\notin$ R y $\Sigma^* \in$ R)

Burlando al halting problem

Si una MT se mueve en un espacio limitado, se puede detectar cuándo entra en un loop ejecutándola y llevando una cuenta de sus pasos:

Una configuración de una MT tiene: (a) una posición (b) un estado q (c) un contenido

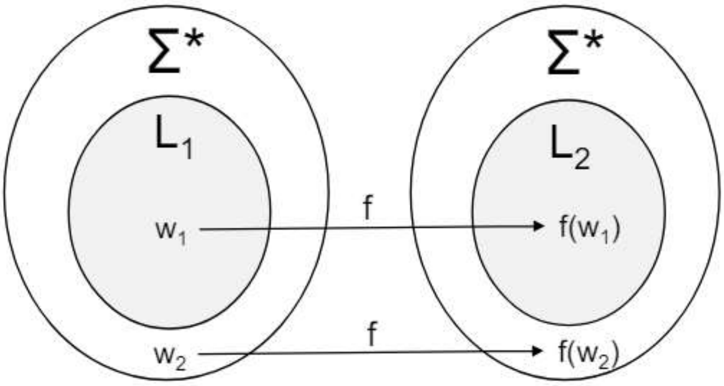
Si M tiene |Q| estados y |Σ| símbolos, antes de repetir una configuración hará a lo sumo:

$x.|Q|.|\Sigma|^x$ pasos (x posiciones, |Q| estados, $|\Sigma|^x$ contenidos).

Reducciones de lenguajes

Dados dos lenguajes L1 y L2 , supongamos que existe una MT Mf que computa una función $f : \Sigma^* \longrightarrow \Sigma^*$ de la siguiente forma:

- a partir de todo $w \in L1$, la MT Mf genera $f(w) \in L2$
- a partir de todo $w \notin L1$, la MT Mf genera $f(w) \notin L2$



La función f es una reducción de L1 a L2. Se anota $L1 \leq L2$, y se dice que la función f es total computable (se computa sobre todas las cadenas).

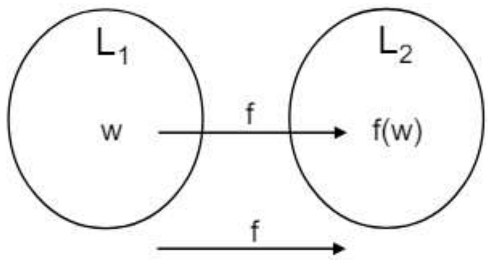
f es computable si existe una MT que la computa y siempre se detiene (realiza n pasos FINITOS). Es decir, para que sea una función de reducción sea válida esta no debe loopear.

La función de identidad mapea cada cadena a sí misma.

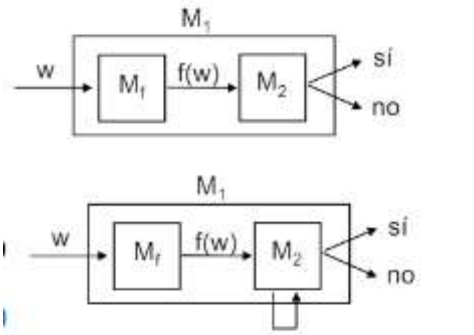
TEOREMA

- Si $L1 \leq L2$ entonces ($L2 \in R \longrightarrow L1 \in R$)
  - Si $L1 \leq L2$ entonces ($L1 \notin R \longrightarrow L2 \notin R$)
- Si $L1 \leq L2$ entonces ($L2 \in RE \longrightarrow L1 \in RE$)
  - Si $L1 \leq L2$ entonces ($L1 \notin RE \longrightarrow L2 \notin RE$)

Es decir, si $L1 \leq L2$: Si $L1 \notin R$, no puede suceder que $L2 \in R$. Si $L1 \notin RE$, no puede suceder que $L2 \in RE$. L2 es tan o más difícil que L1 , resolviendo L2 se resuelve L1.



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

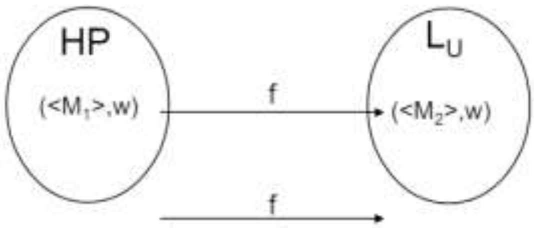


En ambos casos, $w \in L_1$ sii $f(w) \in L_2$
Por eso M_1 responde lo que responde M_2

Ejemplo Reducción para el parcial:

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

<u>Vamos a probar HP ≤ LU :</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 $L_U \notin R$.

<u>Definición de la reducción</u>

f((<M1>, w)) = (<M2>, w), con M2 como M1 , salvo que los estados qR de M1 se cambian en M2 por estados Qa

<u>Computabilidad</u>

Existe una MT Mf que computa f: copia (<M1>, w) pero cambiando los estados qR de M1 por estados qA en M2.

<u>Correctitud</u>

(<M1>, w) ∈ HP ⟶ M1 para sobre w ⟶ M2 acepta w ⟶ (<M2>, w) ∈ LU

(<M1>, w) ∉ HP

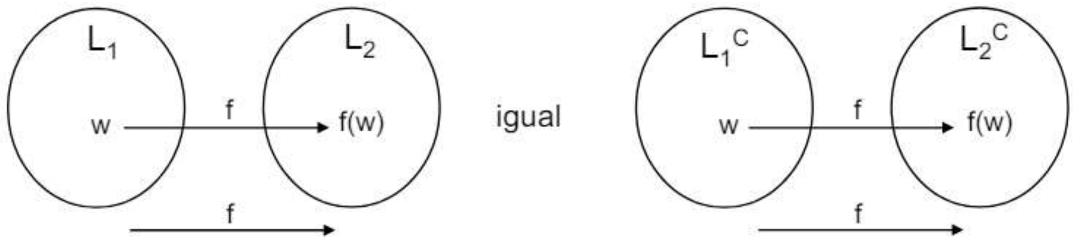
⟶ caso de cadena válida: M1 no para sobre w ⟶ M2 no para sobre w ⟶ (<M2>, w) ∉ LU

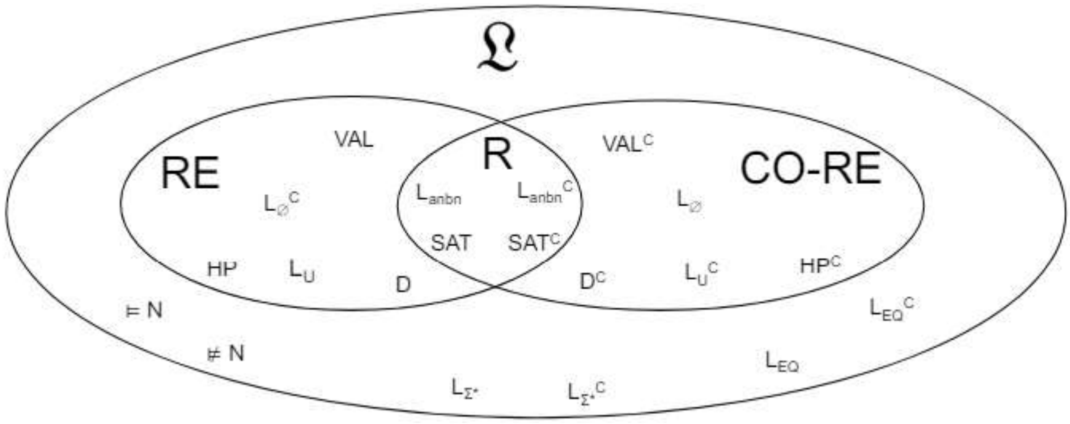
⟶caso de cadena inválida: (<M2>, w) también es una cadena inválida ⟶ (<M2>, w) ∉ LU

Propiedades de las reducciones

- Reflexividad. Para todo lenguaje L se cumple L ≤ L. La función de reducción es la función identidad.
- Transitividad. Si L1 ≤ L2 y L2 ≤ L3 , entonces L1 ≤ L3 .
- Otra propiedad: L1 ≤ L2 sii $L1^C \leq L2^C$. Es la misma función de reducción.

- No se cumple la simetría. L1 ≤ L2 no implica L2 ≤ L.





SAT: fórmulas booleanas satisfactibles.

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

⊨ N: enunciados verdaderos de la aritmética.

LΣ* = { <M>| L(M) = Σ*}

LEQ = {(<M1>, <M2>) | L(M1) = L(M2)}.

Lanbn = {$a^n b^n$ | n ≥ 1}

D = {wi | Mi acepta wi }

D^C = {wi | Mi rechaza wi }

HP = {(<M>, w) | M para sobre w}

HP^C = {(<M>, w) | M no para sobre w}

Todo lenguaje L de RE se puede enumerar:

Sea M1 una MT que acepta L. Vamos a construir una MT M2 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 M1 . Si M1 acepta, imprimir la cadena.

4. Hacer n := n + 1 y volver al paso 2.

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

Sea M1 una MT que decide L. Vamos a construir una MT M2 que genera L en el orden canónico:

1. Generar la primera cadena en el orden canónico.

2. Ejecutar M1 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.

En la clase R se cumple, sin considerar los lenguajes especiales Σ* y ∅, que cualquier lenguaje L1 se puede reducir a cualquier lenguaje L2 . En otras palabras, todos los lenguajes de R tienen la misma dificultad. La prueba es la siguiente:

Sean L1 y L2 distintos de Σ* y ∅. Sean a ∈ L2 y b ∉ L2 . Sean M1 y M2 tales que deciden L1 y L2 .

Definición de la reducción

f(w) = a si w ∈ L1

f(w) = b si w ∉ L1

Computabilidad

Dada w, la MT Mf que computa f ejecuta M1 sobre w, si acepta imprime a y si rechaza imprime b.

Correctitud

Claramente, w ∈ L1 sii f(w) ∈ L2