Clase 4- Rreducciones

martes, 1 de abril de 2025 19:03

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.

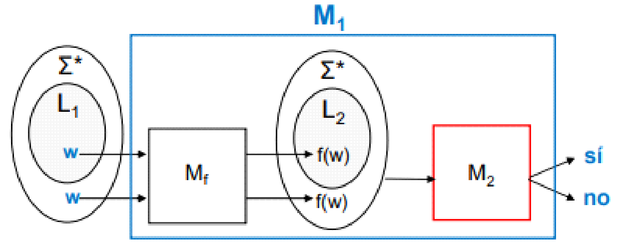
Reducir: derivo un problema a otro. Busco uno maás fácil que a tenga solución y parto de ahi para resolver. Es como usar una subrutina

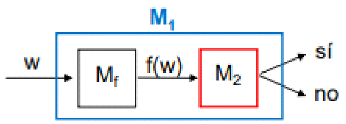
Reducción:
Es una funcio'n total computable que va de todas las cadenas a todas las cadenas. Osea que está definida para todas las cadenas.

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

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

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.

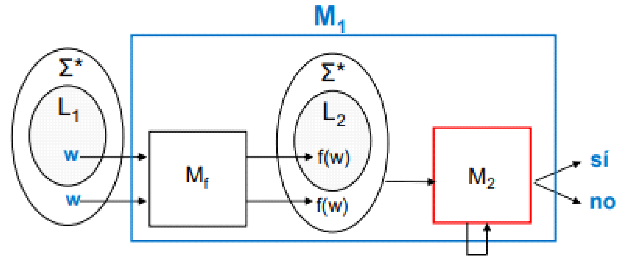
- En lugar de construir de cero una MT M_1 para decidir L_1, se la puede construir a partir de una MT M_2 **conocida** que decide L_2 (noción de invocación a una subrutina en programación).

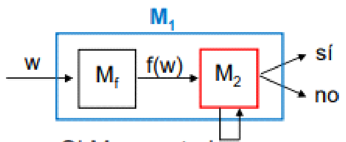
Formalmente: **si $L_1 \leq L_2$, entonces $L_2 \in R \rightarrow L_1 \in R$**
O lo mismo: **si $L_1 \leq L_2$, entonces $L_1 \notin R \rightarrow L_2 \notin R$**

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.

L2 es más diicil que el otro.

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

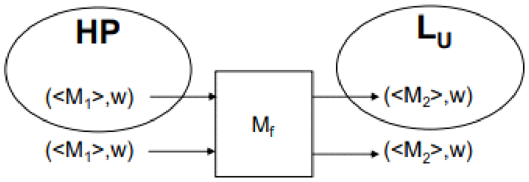
Encontrando una reducción de un lenguaje L_1 que no está en RE a un lenguaje L_2, se prueba que L_2 tampoco está en RE.

Por definición de reducción m1 no puede loopear (creo)
Todo esto parte de que conoces L2 y que no pertenece a RE

Tengo que construir la mt y probar correctitud y computtabilidad

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

M1 y m2 son iguales solo que invierte qr y qa
Si m1 no para sombre w no necesariamente l2 funca, pq puede loopear

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.

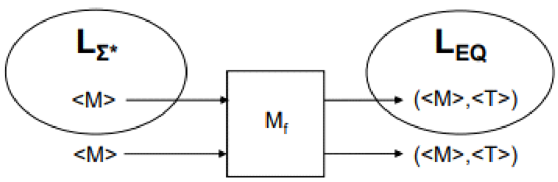
Correctitud:

(<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 loop

Ejemplo 2: ver si un lenguaje acepta todos los lenguajes del ulierso
Leq ver si las maquinas on iguales

L_{Σ^*} = {<M> | L(M) = Σ*} y L_{EQ} = {(<M_1>, <M_2>) | L(M_1) = L(M_2)}.

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



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

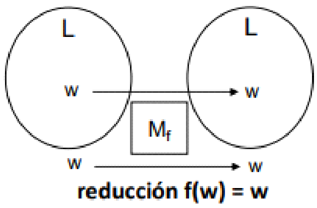
<M> ∈ L_{Σ^*} ⟶ L(M) = Σ* ⟶ L(M) = L(T) ⟶ (<M>, <T>) ∈ L_{EQ}
<M> ∉ L_{Σ^*} ⟶ si <M> es una cadena válida:
L(M) ≠ Σ* ⟶ L(M) ≠ L(T) ⟶ (<M>, <T>) ∉ L_{EQ}
si <M> no es una cadena válida:
(<M>, <T>) tampoco es una cadena válida ⟶ (<M>, <T>) ∉ L_{EQ}

Leq está o no en re? Como hago para decir que si si tengo que comparar cadenas infinitas. Tengo que buscar uno fuera de re y i uedo hacer la reucción pude hacer la reducción.

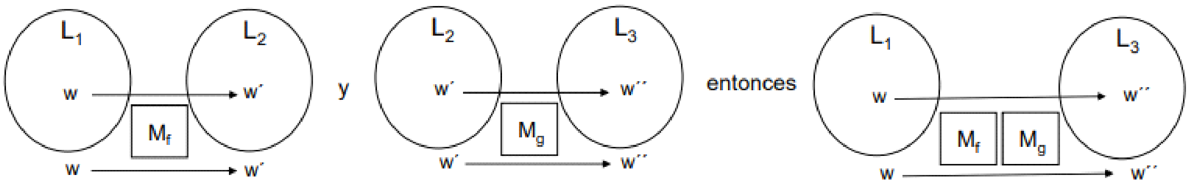
M1 copia m y concatena T. copiar lo que llegue como 1 y ponerle leq.
La reducción funciona pq

Algunas propiedades de las reducciones

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

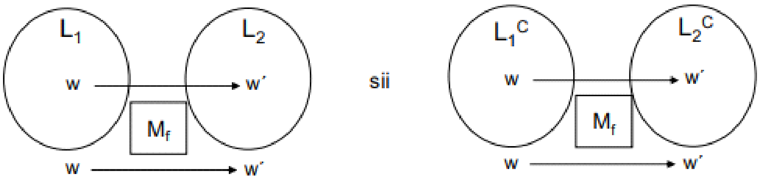


- **Transitividad.** Si $L_1 \leq L_2$ y $L_2 \leq L_3$, entonces $L_1 \leq L_3$.



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

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



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

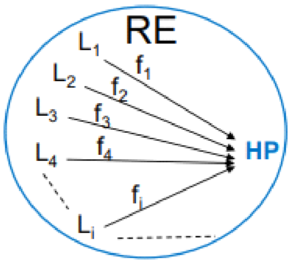
08

Reflex: puedo reducir cualquier mt a si misma
Transitividad: primero aplico f y despues g. consigo h.

Si L1 ○ L2 , entonces L2 es tan o más difícil que L1
Si L1 ∉ R, no puede suceder que L2 ∈ R
Si L1 ∉ RE, no puede suceder que L2 ∈ RE

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 $L \leq HP$:**



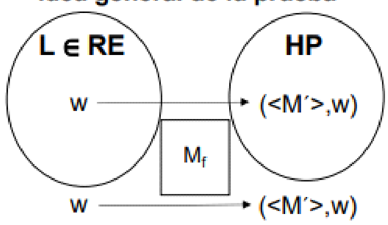
HP es tan o más difícil que cualquier lenguaje de RE.

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.

Si hay un lenguaje en re es que hay una mt que lo acepta
Agarro un lenguaje de re. L por definición tiene un lenguaje que loa cepta. Entonces lo que hace es
Lero lero
Necesito que a codigo de la maquina que acepta l le digo que en lgar de fallar loopee

MT restringidas
Autómata finito: mt muy tonta con una sola cinta solo de lectura. La mt va solo a la derecha siempre
Cuando llega a un blanco para y va a parar si el estado está dentro de un estado final

Ecplicaión práctica

r<re-r<|-re

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

: si L1 ○ L2 y L1 ∉ R, entonces L2 ∉ R. Asi, hay que encontrar una reducción de la forma L1 ○ $L\Sigma^*$, de modo tal que L1 ∉ R. Elegimos como L1 el lenguaje LU.
Quiero buscar un lenguaje m(as dif(icil que el que quier que no esgt(e en r
LU recibe pares> mt, w. salida es una maquina
Tiene que un elemento de mu mandarlo a un elemento de lsigma estrella
Lsigma estrella es una mt ue dice que s(I para todo
Si est(a por fuera de lu tiene que ir afuera de lsigma estrella
Simetr(ia de las reducci(ones> que se reduzcan mutuamente es rer, no suele pasar salvo dentro de r.
Todos los lenguajes de r tienen la misma dificultad

Puedo saber si I cadena est(a en r1 o r2 porque puedo ejecutar la mt en mi mt geeral. Ejecuta m1 sobre w y si me dicce qa tiene que escribir cualquier w que pertenezca a l2 *(hardodeado si quiero(

Ejecuter m1 sobre mt y si me dice que si\

Imprimo una cadena de l2
Si no pertenece, imprio una que no pertenezzca.
No funciona con sigma estrella y vacio
Si quiero reducir de l1 sigma estrella no tengo nada para escribir que no pertenezca a sigma estrella. Con el vacio es lo contrario, no tengo nada para escribir si acepta..
Sigma estrella y vacio sojn com a(un m(as faciles que r

Por qu(e algo no es una reducci(on>
1-pq no es total computable ej en algun momento puede loopear
2- Si l1 acepta y en l2 queda afuera idk no entend(I