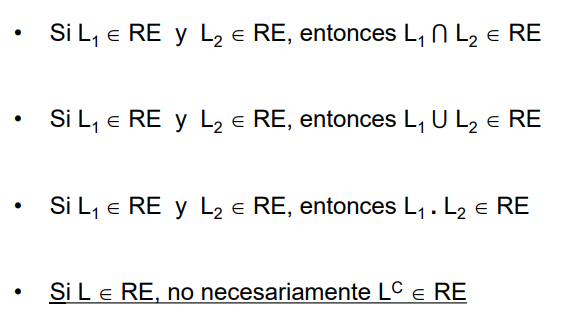
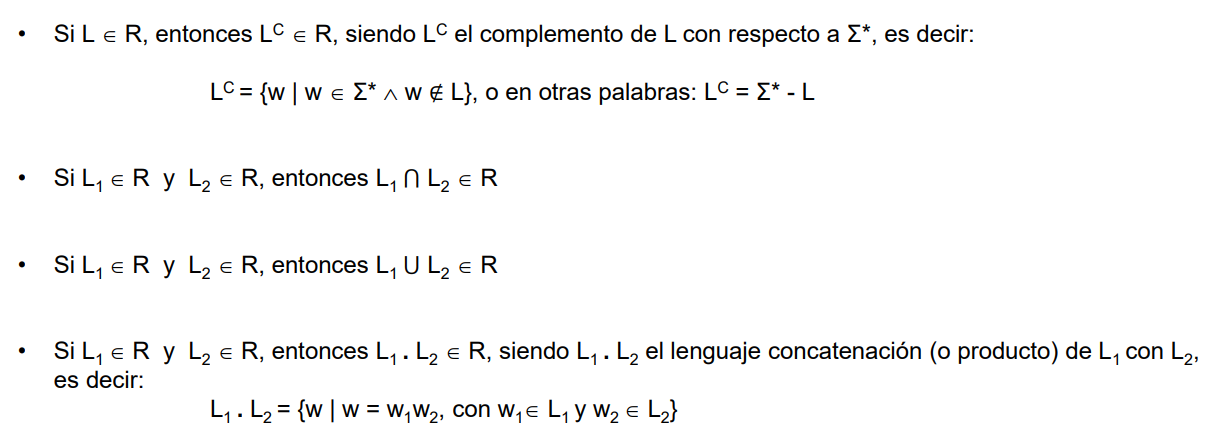
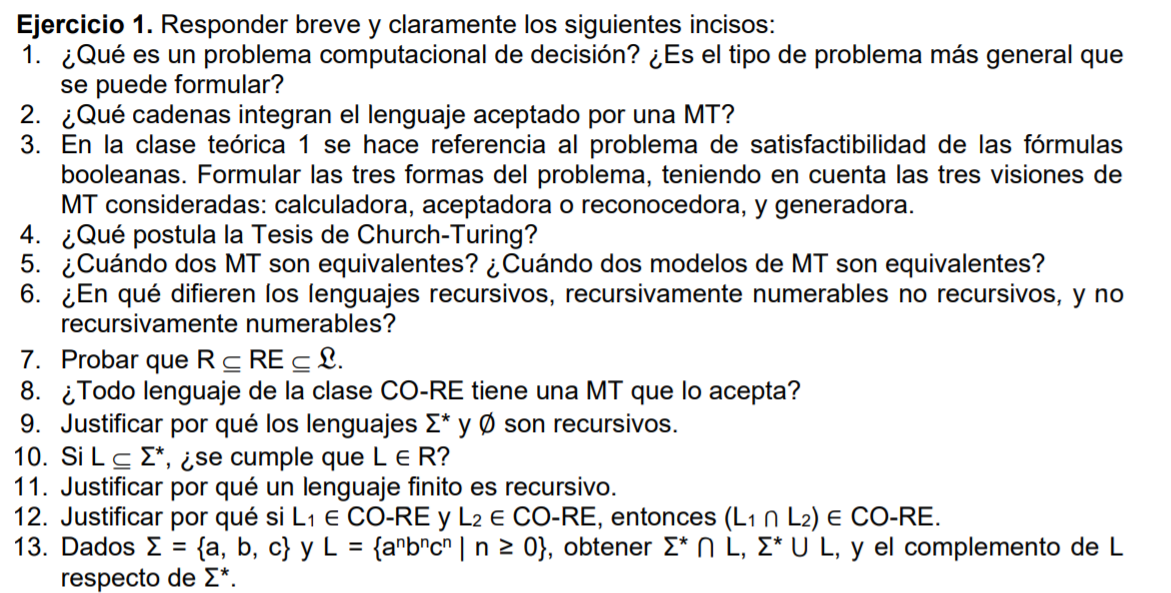
ͰλϵδΣΓ∀⊂⊆⇒ℒαΩΘ

**Propiedades de RE**

****

**Propiedades de R**

****



1. Un problema computacional de decisión es un tipo de problema en el cual se da una respuesta por sí o no. Específicamente, se determina si el string que se recibió, pertenece al conjunto de palabras que la MT reconoce.

No, el más general es el problema computacional de búsqueda, en el que, dado un input, nos devuelve el procesamiento de este como un output.

1. Las cadenas que integran el lenguaje aceptado por una MT son las que la máquina reconoce (esto se hace cuando paran en qA)
2. Una maquina aceptadora es aquella que utiliza una MT, que para un string dado, determine si existe una combinación de valores en sus variables en la que obtengamos como resultado verdadero (que sea satisfactible). Entonces, tenemos que construir una máquina que pruebe todas las combinaciones de valores de verdad en la fórmula. En caso de que alguna combinación satisfaga la fórmula, la máquina se para en qA, si ninguna lo hace, para en qR. La máquina siempre para porque a lo sumo las combinaciones de valores de verdad van a ser 2^n (n es el número de variables de la fórmula)

Una maquina calculadora es aquella que utiliza una MT, que para un string dado, determine si hay una combinación de valores en sus variables en la que obtengamos como resultado verdadero e imprima la combinación en la cinta que actúa como output. Cuando se encuentre una cadena que pueda ser satisfecha, escribe en la cinta de output la asignación de valores que satisfacen la cadena, si no existe una combinación que la satisfaga se deja la cinta en blanco.

Una maquina generadora es aquella que utiliza una MT que genere todas las fórmulas booleanas satisfactibles, entonces se necesita crear todas las cadenas booleanas posibles, y en cada una evaluar lo siguiente:

Si la fórmula es rechazada, se genera la próxima y se vuelve a analizar.

Si la fórmula es aceptada, la escribe en la cadena de output y después se pone un símbolo para diferenciarla del siguiente string. Se debería seguir un orden a la hora de generarlas para asegurar de que no se repitan las formulas analizadas.

1. La Tesis de Church-Turing plantea que todo lo computable puede ser resuelto por una MT
2. Dos MT son equivalentes cuando reconocen el mismo lenguaje. Dos modelos de MT son equivalentes cuando para cada MT de un modelo se puede encontrar una MT' equivalente del otro.
3. Los lenguajes recursivos son aquellos en los que la MT va a parar siempre, en los recursivamente numerables la MT puede parar o quedarse loopeando, los no recursivos son los que no se pueden decidir con una MT que siempre pare, y los no recursivamente numerables son lenguajes que no tienen una MT que lo acepte.
4. *R ⊆ RE* se demuestra por definición, la definición de RE es igual que la de R pero con una restricción menos (la que permite a la MT quedarse loopeando).

*RE ⊆ 𝔏* es demostrado por definición, ya que 𝔏 es el conjunto de todos los lenguajes, por ende cualquier conjunto de lenguajes está incluido en él, entre otros RE.

1. No, solo se podrían aceptar los lenguajes que pertenecen a R dentro del CO-RE
2. Son recursivos porque se puede crear una MT que los reconozca y acepte siempre. Para Ʃ\* hay que hacer una MT que acepte cualquier string, y para ∅ hay que hacer una MT que rechace siempre
3. Esto es falso, porque todos los lenguajes L ⊆ Ʃ\* pero no todos pertenecen a R (como el Halting Problem)
4. Un lenguaje finito es recursivo porque se puede hacer una MT que pare siempre, ya que, para cualquier lenguaje finito, se podría hacer una MT con un estado por carácter y con eso se pueden reconocer todos los strings.
5. Siendo que: *L1 ∈ CO-RE y L2 ∈ CO-RE*

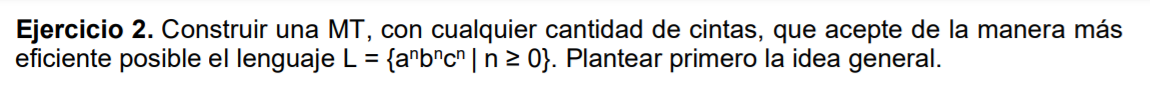
Por la definición de CO-RE L1C ∈ RE *y* L2C ∈ RE y la segunda cláusula de RE (si ambos están en RE, su unión está en RE)

L1C U L2C ∈ RE por las leyes de Morgan L1C U L2C ∈ RE entonces (L1 ∩ L2)C ∈ RE y nuevamente por definición de CO-RE si (L1 ∩ L2)C ∈ RE entonces L1 ∩ L2 ∈ CO-RE.

1. Ʃ\* Ո L = L

Ʃ\* U L = Ʃ\*

El complemento de L respecto a Ʃ\* es LC, o se podría ver como Ʃ\* - L.



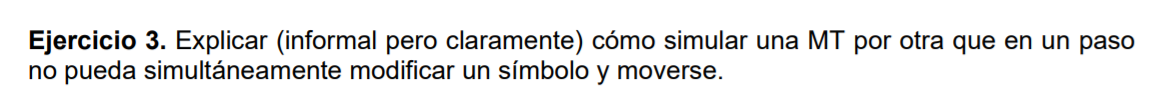
Idea general:

Dada una MT ML que acepta L y para siempre (hipótesis)

Construcción:

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

1. Escribe en unario la cantidad de “a” que hay en el input w sobre la cinta 2.
2. Recorre la cinta 2 controlando que haya igual cantidad de “b” que de unos, en caso de no cumplirse para en qR.
3. Repite el paso anterior controlando las “c”, en caso de cumplirse para en qA y en caso contrario en qR



Habría que hacerlo en 2 pasos, en el primero modificar el símbolo y en el siguiente hacer el movimiento requerido

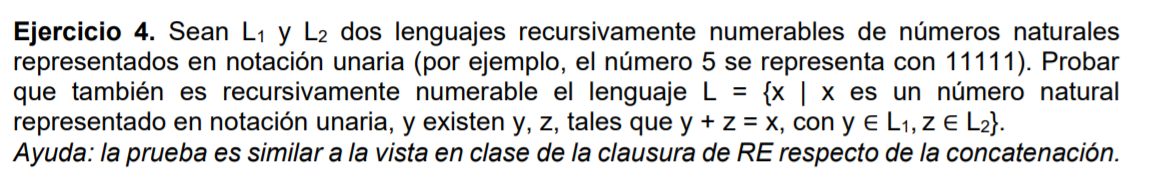
Un ejemplo sería:

δ(q0, 0) = (q1, 1, L)

Pasaría a:

δ(q0, 0) = (q1, 1, S)

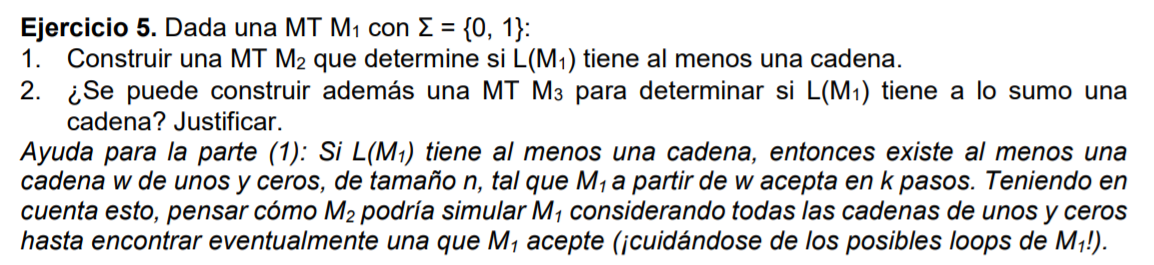
δ(q1, 1) = (q1, 1, L)



Al estar los números representados en notación unaria, se puede pensar la suma como una concatenación por ejemplo, 11 + 111 = 11111 y 11 . 111 = 11111, entonces se puede pensar en que x = yz.

En este ejemplo la forma de probarlo es con la clausura de RE con w tomando el valor de x, w2 el de z y w1 el de y.

Si L1 ∈ RE y L2 ∈ RE, entonces L1 . L2 ∈ RE, siendo L1 . L2 el lenguaje concatenación (o producto) de L1 con L2 , es decir: L1 . L2 = {w | w = w1w2 , con w1∈ L1 y w2 ∈ L2 }

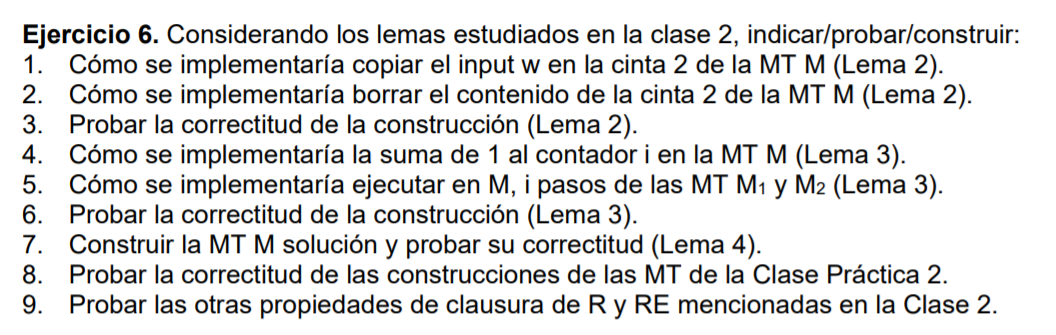


* + - 1. Los pasos para la construcción de esta máquina serían:

1. Hacer i := 1.
2. Copiar i cadenas en una cinta y hacer i pasos en cada una, si encuentra la cadena para en qA.
3. i:=i+1
4. Vuelve al paso 2

De esta manera la maquina buscara de manera indefinida hasta que se determine si L(M1) tiene al menos una cadena.

* + - 1. No se puede porque para saber si acepta solo 1, hay que probar con el conjunto de todos los strings posibles (infinito)



1. Primero se deberia plantear un estado que inserte en la cinta 2 lo mismo que lee en la cinta 1 (qC por ejemplo) hasta que encuentre un blanco. Habria que hacer una funcion de transicion por cada combinacion de caracteres del alfabeto.

(\* significa cualquier caracter)

Por ejemplo

δ(qC, a, \*) = (qC, (a, D), (a, D))

δ(qC, b, \*) = (qC, (b, D), (b, D))

…

1. Hay que tener en cuenta, que la cinta 2 puede no estar en el inicio por lo que habria que ponerla al inicio y despues, plantear un estado qD, que mientras lea caracteres en la cinta 2, pegue un blanco (habiendo cualquier cosa en la cinta 1)

por ejemplo

δ(qD, a, \*) = (qD, (a, S), (B, D))

δ(qD, b, \*) = (qD, (b, S), (B, D))

...

1. Sabiendo que L(M1) = L1 | L(M2) = L2

Hay que probar que: L(M) = L1 ⋂ L2

Entonces partiendo de un x ϵ L1 ⋂ L2, podemos saber que x ϵ L1 y x ϵ L2

Si x ϵ L1 y x ϵ L2 entonces M1 y M2 lo aceptan

si M1 y M2 lo aceptan entonces M lo aceptan por ende x ϵ L(M)

probando la inversa

si x ϵ L(M) entonces M1 y M2 la aceptan por ende x ϵ L1 ⋂ L2

1. Habria que plantear un estado que mientras lea 1 en la cinta 4, mueva a la derecha y una vez encuentre el blanco, inserte un 1.
2. Una vez que la cinta 4 apunta al principio, habria que plantear un estado que cada vez que la cinta 4 lee un 1, se ejecuta un paso de M1 y M2.
3. Sabiendo que L1 = L(M1) | L2 = L(M2)

probar que L(M) = L1 ⋃ L2

partiendo de x ϵ L1 ⋃ L2, sabemos que x ϵ L1 v x ϵ L2

entonces M1 o M2 lo van a aceptar

y si alguna de las maquinas lo acepta, M lo acepta, por ende, x ϵ L(M)

probando la inversa

si x ϵ L(M) entonces M1 o M2 la aceptan por ende x ϵ L1⋃ L2

1. M tiene 4 cintas. En la cinta 1 tiene el input w. En las cintas 2 y 3 ejecuta ML y MLC . En la cinta 4 tiene un contador i de pasos. La MT hace:
   1. Copia w en las cintas 2 y 3, y en la cinta 4 hace i := 1.
   2. Ejecuta a lo sumo i pasos de ML sobre w en la cinta 2, y a lo sumo i pasos de MLC sobre w en la cinta 3. Si ML acepta, M acepta y si MLC acepta, M rechaza
   3. Borra el contenido de las cintas 2 y 3, copia w en ellas desde la cinta 1, suma 1 a i en la cinta 4, y vuelve al paso 2
2. 1. El lenguaje L1 ⦁ L2 contiene todas las cadenas w = w1w2 , tales que w1ϵ L1 y w2 ϵ L2

Probar que L(M) = L1 ⦁ L2

si x ϵ L1 ⦁ L2 => x = w1w2

en n iteraciones, siendo n menor a la longitud de w1w2, la maquina ejecutara w1 en M1 y w2 en M2

y como w1 ϵ L(M1) y w2 ϵ L(M2) por lo tanto x ϵ L(M).

probar la inversa

si x ϵ L(M) => x = w1w2

y como w1ϵ L1 y w2 ϵ L2 => x ϵ L1 ⦁ L2

* 1. El lenguaje L1 ⦁ L2 contiene todas las cadenas w = w1w2 , tales que w1ϵ L1 y w2 ϵ L2

Probar que L(M) = L1 ⦁ L2

si x ϵ L1 ⦁ L2 => x = w1w2

en n iteraciones y k pasos, siendo n menor a la longitud de w1w2, k una cantidad finita de pasos, la maquina ejecutara w1 en M1 y w2 en M2 (las maquinas ejecutaran k veces w1 y w2)

y como w1 ϵ L(M1) y w2 ϵ L(M2) por lo tanto x ϵ L(M).

probar la inversa

si x ϵ L(M) => x = w1w2

y como w1ϵ L1 y w2 ϵ L2 => x ϵ L1 ⦁ L2

esto es rarisimo

* 1. probar que L(M) = L1 U L2

si hay un x ϵ L1 U L2  => x ϵ L1 v x ϵ L2

entonces como la maquina M ejecuta tanto M1 (que acepta L1) y M2 (que acepta L2) y como una de las dos acepta si o si, M va a aceptar

probar la inversa

si hay x ϵ L(M) => x pertenece a x ϵ L1 v x ϵ L2

entonces x ϵ L1 U L2

1. 1. Complemento de R, probar que si L ϵ R => Lc ϵ R

L(M) = Lc

L(M’) = L

sea un x ϵ Lc  entonces la maquina M lo va a aceptar porque M’ lo va a rechazar

sea un x ϵ L entonces la maquina M lo va a rechazar porque M’ lo va a aceptar

inversa

si x ϵ L(M) entonces la maquina M’ lo rechaza por ende x ϵ Lc