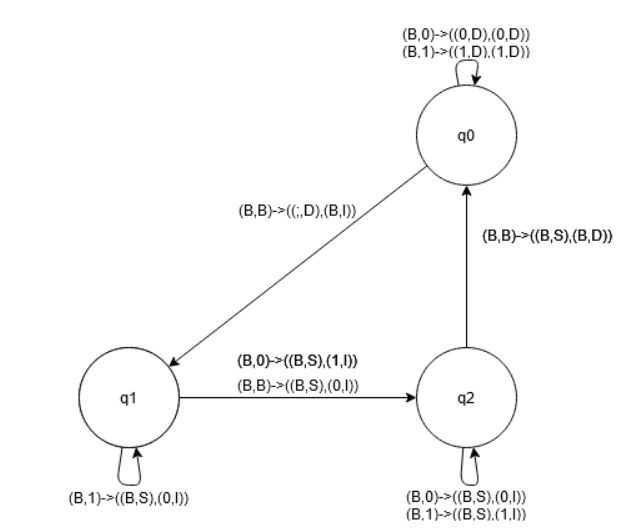
1) Construir una máquina de Turing que escriba en la primera cinta las palabras de

{0,1}\* en orden canónico separadas por un símbolo ";". Obviamente esta máquina

nunca se detiene.

De hecho, fue más fácil de lo que pensé que sería.



2) Sean Σ ={a,b} y L el conjunto de todos los lenguajes definidos sobre Σ. Diga si las

siguientes afirmaciones son verdaderas o falsas:

1. L – R = ∅

Falso. No todos los lenguajes son recursivos enumerables, lo cual es muy importante, ya que indica que no todos los lenguajes son computables.

1. RE – R ≠ ∅

Verdadero. Hay lenguajes para los que es imposible hacer que MTs se detengan en todas las entradas.

1. {λ} ∈ (L – CO-RE )

Acá todo depende si {λ} pertenece a CO-RE, ya que sabemos que {λ} pertenece a L .

Para que {λ} pertenezca a CO-RE entonces complemento de {λ} tiene que pertenecer a RE.

El complemento de {λ} son todos los elementos de Σ\* que no pertenecen a {λ}, llamémosle c{λ}.

Para que c{λ} pertenezca a RE entonces para c{λ} debe existir una MT que para todos los elementos de c{λ} los acepte.

¿Esta máquina existe?

La respuesta evidentemente es que si, puedes simplemente crear una máquina que espere cadenas de símbolos “a” y símbolos “b”, pero no acepta que la cadena no tenga símbolos, sus transiciones serían:

d(q0,B)=(qR,B,S)

d(q0,a)=(q1,a,D)

d(q0,b)=(q1,b,D)

d(q1,a)=(q1,a,D)

d(q1,b)=(q1,b,D)

d(q1,B)=(qA,B,D)

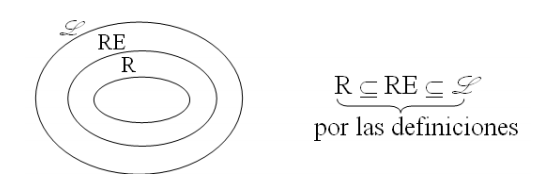
.: La MT existe.

.: c{λ} pertenece a RE.

.: {λ} pertenece a CO-RE.

.: La proposición es Falsa.

1. RE ∪ R = L



Segun la teoria L – RE = ∅, además R no puede ser mas grande que RE, ni este que L. Por lo

tanto no L incluye de forma estricta a RE.

.: Es imposible que la proposición sea verdadera, ya que RE ∪ R está incluido estrictamente

en L.

1. Σ\* ∈ R

Recordemos que:

Σ\*={λ,a,b,aa,ab,ba,bb,aaa,aab,aba,...}

L=p(Σ\*)

Entonces es natural pensar que Σ\* ∈ L ya que Σ\* ∈ p(Σ\*).

Para que Σ\* ∈ R debe existir una MT que acepte todas las frases pertenecientes a Σ\* y rechace las que no pertenecen a este, y siempre deteniéndose en algún punto, esta máquina de turing la puedo definir como:

d(q0,a)=(q0,a,D)

d(q0,b)=(q0,b,D)

d(q0,B)=(qA,B,D)

.: Ya que la MT existe, entonces la proposición es verdadera

1. ∅ ∈ RE

Puede marear un poco este enunciado, pero si lo pensamos nos daremos cuenta que un lenguaje que acepte ∅ sería un lenguaje que no acepte ninguna entrada, para los requisitos de RE lo que necesitamos seria o una máquina que para toda entrada vaya al qR, una que para toda entrada entre a un loop infinito, o una máquina híbrida que para algunos entre en loop y para otros valla a qR.

La 1ra la podemos definir como:

d(q0,x)=(q0,x,S)

La 2da la podemos definir como:

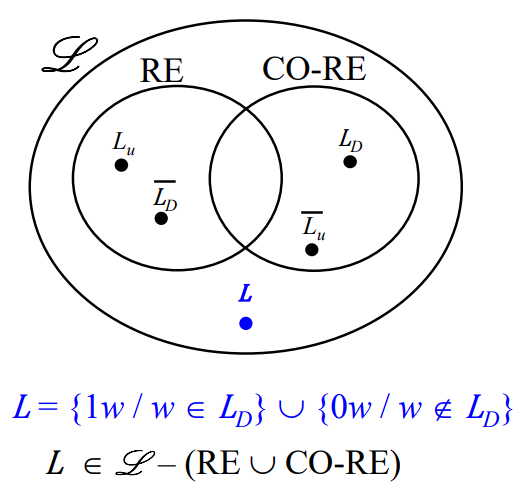
d(q0,x)=(qR,x,S)

La tercera realmente no es necesario definirla.

Ya con una máquina definida hemos probado que ∅ ∈ RE, ya que no solo hemos probado que existe una MT que acepta ∅, sino 2.

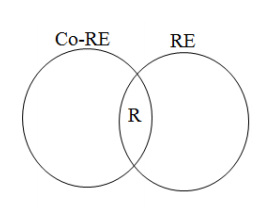
.: La proposición es Verdadera.

1. CO-RE = RE



Falso.

1. (L – RE) = CO-RE



Sabemos que RE intersección CO-RE = R, por lo tanto al quitarle al lenguaje RE, no queda CO-RE, sino una parte de este, y elementos de L que no son de CO-RE.

.: La proposición es falsa.

1. ab ∈ Σ\*

Recordemos que:

Σ\*={λ,a,b,aa,ab,ba,bb,aaa,aab,aba,...}

.: La proposición es verdadera.

1. CO-R ⊂ CO-RE

L ∈ CO-R (por definición de CO-R)

=> L ∈ R (Teorema 1 de la teoría)

=> L ∈ RE (por definición de R y RE)

=> L ∈ CO-RE (por definición de CO-RE)

.: La proposición es verdadera.

1. a ∈ R

Falso. R es un conjunto de conjuntos de palabras y a no es un conjunto.

1. {a} ∈ RE

Verdadero, se puede definir una máquina que solo acepte {a}, las transiciones serían las siguientes:

d(q0,a)=(q1,a,D)

d(q0,b)=(qR,b,S)

d(q1,B)=(qA,B,S)

d(q1,a)=(qR,a,S)

d(q1,b)=(qR,b,S)

.: La proposición es Verdadera.

3) Si L∈(RE - R)

Primero pensemos que es L.

L es un lenguaje.

L es recursivo enumerable pero no es recursivo, osea que existe una MT que lo acepte pero no se detiene siempre.

podríamos definir:

* L=p(Σ\*)
* Σ={x1,x2,x3,...,xn}

a) ¿Existirá alguna máquina de Turing que rechace parando en qR si su entrada está en L y rechace loopeando si su entrada no está en L?

Pensemos bien que nos está diciendo el enunciado.

Pregunta si existe una MT que cumpla las siguientes condiciones:

* Si su entrada está en L -> para en qR.
* Si su entrada no está el L -> Loopea.

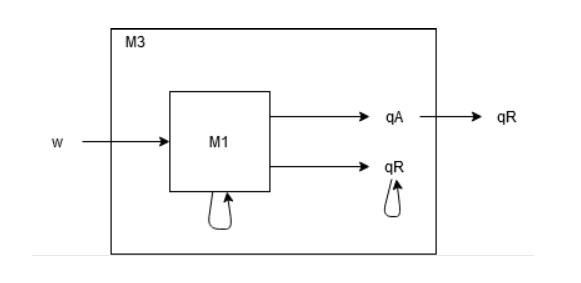
Sabemos que:

* L∈(RE - R)

Si tuviera que desarrollar esta máquina hipotética primero partiría de que sabemos que existe una máquina M1 tal que L=L(M1) pero no se detiene siempre, pero no existe máquina M2 tal que tal que L=L(M2) y se detiene siempre (lo sabemos porque L∈(RE - R)).

Considerando esto vamos a armar una MT llamada M3 tal que para toda entrada w si M1 termina en qA esto significa que w ∈ L y por ende nuestra M3 terminará en qR, para cualquier otra entrada w tal que w ∉ L el enunciado nos pide que la máquina lopee, sabemos que para algunos w ∉ L M1 loopeara y es posible que para otros termine en qR.

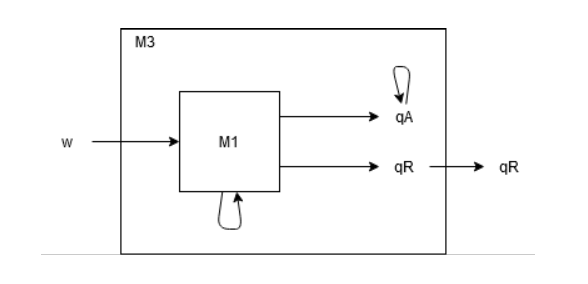
Para los que lopean los dejaremos así, pero también queremos que los que no loopean terminen en loopeo, así que armaremos a M3 como el gráfico indica:



Con esto demostramos que existe como mínimo una MT que rechace parando en qR si su entrada está en L y rechace loopeando si su entrada no está en L.

b) ¿Existirá alguna máquina de Turing que rechace loopeando si su entrada está en L y rechace parando en qR si su entrada no está en L?

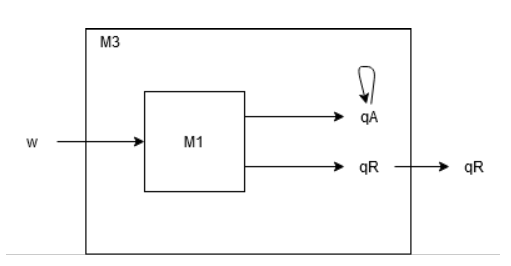
Manteniendo la misma idea que en el inciso anterior armaremos una MT M3 que aproveche la MT M1 para lograr su objetivo



En el gráfico si w ∈ L entonces será aceptada por M1, y M3 hará que loopee, el tema es que si w ∉ L sabemos que necesariamente habrá 1 w para el que M1 loopee, lo que nos hará imposible llevar todos los w ∉ L al estado qR, este es el caso en el ejemplo y también es la razón por la que la MT que pide el enunciado es imposible de armar bajo las condiciones indicadas.

c) De existir, qué lenguaje reconocería esta máquina de Turing.

Un Lenguaje sea L2 ∈ R, este lenguaje siempre se detendrá, por lo que nunca ocurrirá que M1 se quede loopeando por ende siempre se detendrá en qA o en qR, por ende siempre sabremos si w ∈ L o si w ∉ L, podremos hacer que si termina en qA loopee y si termina en qR como esto indica que w ∉ L entonces haremos que pare en qR, sería así:



Por lo tanto, cualquier lenguaje recursivo será reconocido por esta MT.

4) Sea L ={w | Existe alguna Máquina de Turing M que acepta w} ¿L ∈ R? Justifique.

¿Podríamos decir cuál es este lenguaje L?

Creo que no, ya que no se los símbolos que conforman las palabras solo sé que son palabras aceptadas por alguna máquina. Pero podemos realizarnos ciertas preguntas:

¿Si tenemos una palabra/cadena w en particular, podríamos decir si hay una MT que la acepte? ¿cuál?.

Siempre se puede inventar una máquina que para todo símbolo "w" perteneciente a Sigma U {B} haga haga:

delta(q0,w)=(qA,w,S)

con lo cual aceptaría todas las cadenas incluso lambda.

Por lo tanto, con esto tendríamos cuál es el lenguaje L y, en función de eso, podemos responder la pregunta del ejercicio:

¿L pertenece a R?

De esto deducimos que L=Σ\* y L ∈ R

5) Conteste y justifique:

a) ¿L es un conjunto infinito contable?

L será infinito contable si |L |<=|N|

Para que esto ocurra debemos encontrar una funcion inyectiva de L ->N

¿Podemos definir una funcion como esa?

La respuesta es sí.

Primero pensaremos que es L :

Llamamos L al conjunto de todos los lenguajes definidos sobre el alfabeto Σ, es decir: L = p(Σ\*)

¿Qué es Σ?

Σ={x1,x2,...,xn}

¿Qué es Σ\*?

Σ\* en la práctica 2 demostramos que es infinito contable

Σ\*={λ, x1, x2,..., xn, x1x1, x1x2, x2x1, x2x2, x1x1x1, x1x1x2,...,xn...xn}

Una pregunta importante es: ¿hay 2 subconjuntos de Σ\* iguales?

La respuesta es no, podríamos agruparlos de la siguiente manera

p(Σ\*)={{∅},{λ},{x1},{x2},...,{xn},{λ,x1},{λ,x2}...}

Primero tenemos 1 elemento con 0 elementos

n+1 elementos con 1 elemento

luego habría cierta cantidad de elementos con 2 elementos y así.

De esta forma agrupamos los elementos de p(Σ\*) en distintos subconjuntos según su cantidad de elementos.

Si a estos elementos de los elementos de p(Σ\*) los interpretamos bajo un sistema n-esimal entonces podremos a cada uno asignarle un natural.

Una vez que cada uno tenga un natural los podremos definir a cada elemento de p(Σ\*) como una tupla (i,j) siendo i el grupo de elementos al que pertenece (según su cantidad de elementos) y j su posición dentro de este conjunto, la podríamos calcular en base a la suma de los elementos que forman este elemento de p(Σ\*), obviamente a estos ya los habíamos transformado del sistema n-esimal al sistema decimal.

Por ejemplo:

* {∅} ->(0,0)
* {λ}->(1,0)
* {x1}->(1,1)
* {λ,x1}->(2,1)
* {λ,x2}->(2,2)

Muy bien, asi definimos una funcion inyectiva f: L ->(N,N)

También sabemos por prácticas anteriores que existe una funcion inyectiva tal que g: (N,N)->N

Por silogismo podemos definir una nueva funcion tal que h:L ->N

.: Ya que encontramos la funcion inyectiva entonces L es infinito contable

b) ¿RE es un conjunto infinito contable?

RE es un subconjunto de L, por lo tanto |RE|<=|L |, también sabemos que |L |<=|N| (por el inciso a)

.: Por silogismo entonces |RE|<=|N|, por lo tanto RE es infinito contable.

c) ¿L – RE es un conjunto infinito contable?

Misma lógica que en el inciso anterior:

L – RE es un subconjunto de L, por lo tanto |L – RE|<=|L |, también sabemos que |L |<=|N| (por el inciso a).

.: Por silogismo entonces |L – RE|<=|N|, por lo tanto L – RE es infinito contable.

d) Existe algún lenguaje L ∈ L , tal que L sea infinito no contable

Lo estuve pensando y llegue a la conclusión de que:

Sabemos que L es infinito contable, pero eso no implica que los lenguajes que lo forman también lo sean, así que pensé:

Los L son conjuntos que describen una serie de palabras aceptadas, estas palabras tienen un orden canónico así que si hacemos una función que para cada palabra le corresponda su posición en el orden canónico entonces tendríamos lo que necesitamos para demostrar que |L|<=|N| y por ende demostrar que L es infinito contable.

El punto preguntaba por un L que sea infinito no contable.

De esto entiendo que no ya que demostré que todo L es infinito contable.

El ayudante contestó que está respondido en la explicación de los miércoles que le corresponde (14/10 o 21/10), la dejo para ver otro día.

6) Sea L un lenguaje definido sobre Σ. Demostrar que:

a) L ∉ R ⇒ L ∉ R

Esto es cierto.

Sabemos que:

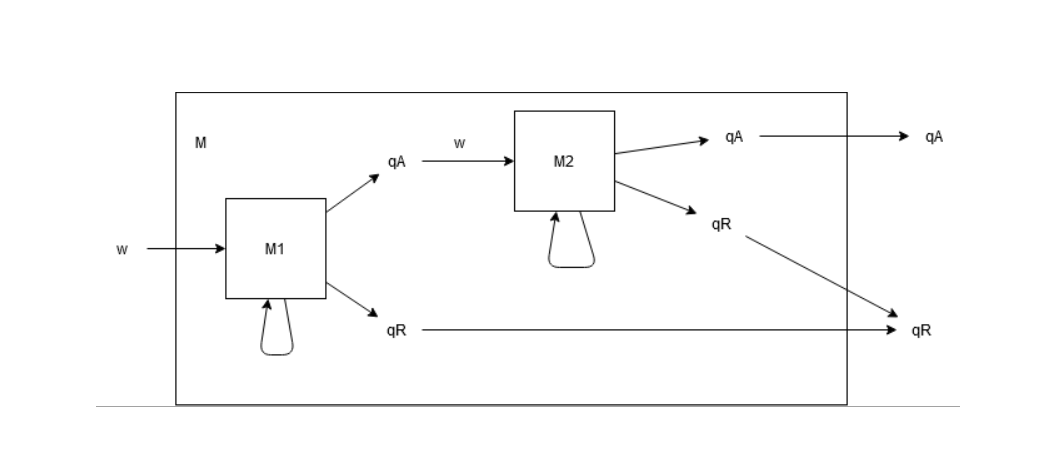
1. R = Co-R
2. L ∈ Co-R sii L ∈ R

Según esto sabemos que si L ∉ R entonces L ∉ Co-R (2), si L ∉ Co-R entonces L ∉ R (1)

.: L ∉ R ⇒ L ∉ R

b) (L1 ∈ RE) AND (L2 ∈ RE) ⇒ L1 ∩ L2 ∈ RE

Sabemos que si un L es RE entonces este lenguaje no necesariamente se detiene, y este lenguaje se detiene en qA para las entradas que acepta.



Definimos M tal que si una alguna de las 2 maquinas termina en qR entonces M termina en qR, si ambas aceptan w entonces M acepta w, si M1 rechaza loopeando o si M1 acepta w pero M2 rechaza loopeando w entonces M loopea.

.: si es cierto que (L1 ∈ RE) AND (L2 ∈ RE) ⇒ L1 ∩ L2 ∈ RE

Demostración:

Sean M1 y M2 MT de una sola cinta tq L1=L(M1) y L2=L(M2).

Se eligen ambas MT tal que se no necesariamente se detienen para toda entrada, seguro existen porque ambos lenguajes pertenecen a RE.

M funciona de la siguiente manera:

1. Copia la entrada en la 2da cinta y posiciona el cabezal en el principio de la entrada en la 2da cinta.
2. Simula M1 sobre la cinta 2. Si M1 para en qR -> M para en qR, si M1 para en qA ir al punto 3, si M1 loopea -> M loopea.
3. Borra la cinta 2.
4. Copia w de la cinta 1 a la cinta 2.
5. Simula M2 sobre la cinta 2. Si M2 para en qR -> M para en qR, si M1 para en qA -> M para en qA, si M2 loopea -> M loopea.

¿Cómo sería concretamente la codificación de la MT?

Sería muy similar a la que se encuentra en la diapositiva 5 de la teoría 9:

M=<Q,Σ,Γ,δ,q0,qA,qR>

δ: Q ∪ Γ2 → Q ∪ {qA,qR} x (Γ x {D,I,S})2

con Q1 ∪ {q1A,q1R} ∪ Q2 ∪ {q2A,q2R} ⊆ Q ; Γ1 ∪ Γ2 ⊆ Γ

δ(qi ,a,b)=(qj,(c,m1),(d,m2)) con qi ∈ Q; qj ∈ Q ∪ {qA,qR}; a,b,c,d ∈ Γ; m1,m2 ∈ {D,I,S}

1. Copia la entrada en la 2da cinta y posiciona el cabezal en el principio de la entrada en la 2da cinta.

δ(q0 ,x,B)=(q0,(x,D),(x,D)) (Vx)(x ∈ Σ) (Copia la entrada en la cinta dos)

δ(q0 ,B,B)=(q1,(B,S),(B,I))

δ(q1 ,B,x)=(q1,(B,S),(x,I)) (Vx)(x ∈ Σ) (Se dirige al inicio de la cinta 2)

δ(q0 ,B,B)=(q1,(B,S),(B,D))

1. Simula M1 sobre la cinta 2. Si M1 para en qR -> M para en qR, si M1 para en qA ir al punto 3, si M1 loopea -> M loopea.

Para cada δ1(qi ,x)=(qj,y,m) se define δ(qi1,B,x)=(qj1,(B,S),(y,m))

(Simulación en cinta 2, nótese que si la simulación de M1 loopea entonces M loopeara)

δ(qR1,B,x)=(qR,(B,S),(x,S)) (Vx)(x ∈ Γ1)

(Si M1 rechaza, M también)

δ(qA1,B,x)=(q3,(B,S),(x,S)) (Vx)(x ∈ Γ1)

(Si M1 acepta ir al punto 3)

Un detalle no menor es que cada vez que para cada vez que M1 debe usar B entonces M1 usa #

Esto es así para facilitar el punto 3.

1. Borra la cinta 2.

Ya sabemos que la M2 no uso B así que no quedaron espacios vacíos en la cinta. Sabemos que cuando acabamos estamos en un símbolo x de la cinta, lo que harán las transiciones que voy a definir es: moverse a derecha borrar hasta blanco, se mueve a Izquierda hasta encontrar el símbolo x que encontramos antes y luego borramos hacia izquierda.

δ(q3,B,x)=(q3.1,(B,S),(x,D)) (Vx)(x ∈ Γ1)

δ(q3.1,B,x)=(q3.2,(B,S),(B,D)) (Vx)(x ∈ Γ1 - {B})

δ(q3.2,B,B)=(q3.2,(B,S),(B,I))

δ(q3.2,B,x)=(q3.3,(B,S),(B,I)) (Vx)(x ∈ Γ1 - {B})

δ(q3.3,B,x)=(q3.3,(B,S),(B,I)) (Vx)(x ∈ Γ1 - {B})

δ(q3.3,B,B)=(q4,(B,S),(B,S)) (Cinta 2 borrada vamos al punto 4)

1. Copia w de la cinta 1 a la cinta 2.

δ(q4 ,B,B)=(q4.1,(B,I),(B,S))

δ(q4.1 ,x,B)=(q4.2,(x,I),(x,I)) (Vx)(x ∈ Σ) (Copia la entrada en la cinta dos)

δ(q4.2 ,B,B)=(q02,(B,S),(B,D))

1. Simula M2 sobre la cinta 2. Si M2 para en qR -> M para en qR, si M1 para en qA -> M para en qA, si M2 loopea -> M loopea.

Para cada δ2(qi ,x)=(qj,y,m) se define δ(qi2,B,x)=(qj2,(B,S),(y,m))

(Simulación en cinta 2, nótese que si la simulación de M1 loopea entonces M loopeara)

δ(qR2,B,x)=(qR,(B,S),(x,S)) (Vx)(x ∈ Γ2)

(Si M2 rechaza, M también)

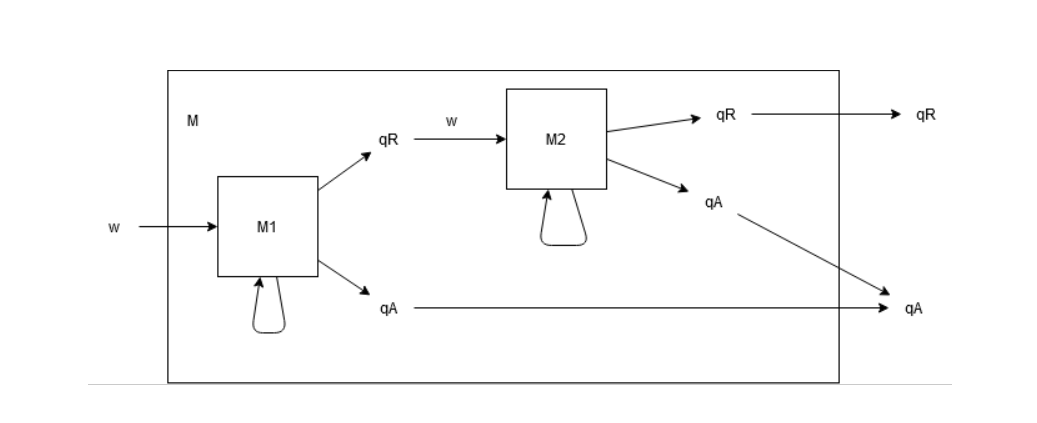
δ(qA2,B,x)=(q5,(B,S),(x,S)) (Vx)(x ∈ Γ2)

(Si M2 acepta, M también)

Esta simulacion si puede usar B ya que luego no tendrá que borrar la cinta

c) (L1 ∈ RE) OR (L2 ∈ RE) ⇒ L1 ∪ L2 ∈ RE

~~Armamos una MT M similar a como lo hicimos en el caso anterior, con algunas diferencias, el gráfico sería este:~~



~~M funciona de la siguiente manera:~~

1. Copia la entrada en la 2da cinta y posiciona el cabezal en el principio de la entrada en la 2da cinta.
2. Simula M1 sobre la cinta 2. Si M1 para en qA -> M para en qA, si M1 para en qR ir al punto 3, si M1 loopea -> M loopea.
3. Borra la cinta 2.
4. Copia w de la cinta 1 a la cinta 2.
5. Simula M2 sobre la cinta 2. Si M2 para en qA -> M para en qA, si M1 para en qR -> M para en qR, si M2 loopea -> M loopea.

¿Cómo sería concretamente la codificación de la MT?

Sería muy similar a la que se encuentra en la diapositiva 5 de la teoría 9:

M=<Q,Σ,Γ,δ,q0,qA,qR>

δ: Q ∪ Γ2 → Q ∪ {qA,qR} x (Γ x {D,I,S})2

con Q1 ∪ {q1A,q1R} ∪ Q2 ∪ {q2A,q2R} ⊆ Q ; Γ1 ∪ Γ2 ⊆ Γ

δ(qi ,a,b)=(qj,(c,m1),(d,m2)) con qi ∈ Q; qj ∈ Q ∪ {qA,qR}; a,b,c,d ∈ Γ; m1,m2 ∈ {D,I,S}

1. Copia la entrada en la 2da cinta y posiciona el cabezal en el principio de la entrada en la 2da cinta.

δ(q0 ,x,B)=(q0,(x,D),(x,D)) (Vx)(x ∈ Σ) (Copia la entrada en la cinta dos)

δ(q0 ,B,B)=(q1,(B,S),(B,I))

δ(q1 ,B,x)=(q1,(B,S),(x,I)) (Vx)(x ∈ Σ) (Se dirige al inicio de la cinta 2)

δ(q0 ,B,B)=(q1,(B,S),(B,D))

1. Simula M1 sobre la cinta 2. Si M1 para en qR -> M para en qR, si M1 para en qA ir al punto 3, si M1 loopea -> M loopea.

Para cada δ1(qi ,x)=(qj,y,m) se define δ(qi1,B,x)=(qj1,(B,S),(y,m))

(Simulación en cinta 2, nótese que si la simulación de M1 loopea entonces M loopeara)

δ(qA1,B,x)=(qA,(B,S),(x,S)) (Vx)(x ∈ Γ1)

(Si M1 acepta, M también)

δ(qR1,B,x)=(q3,(B,S),(x,S)) (Vx)(x ∈ Γ1)

(Si M1 rechaza ir al punto 3)

Un detalle no menor es que cada vez que para cada vez que M1 debe usar B entonces M1 usa #

Esto es así para facilitar el punto 3.

1. Borra la cinta 2.

Ya sabemos que la M2 no uso B así que no quedaron espacios vacíos en la cinta. Sabemos que cuando acabamos estamos en un símbolo x de la cinta, lo que harán las transiciones que voy a definir es: moverse a derecha borrar hasta blanco, se mueve a Izquierda hasta encontrar el símbolo x que encontramos antes y luego borramos hacia izquierda.

δ(q3,B,x)=(q3.1,(B,S),(x,D)) (Vx)(x ∈ Γ1)

δ(q3.1,B,x)=(q3.2,(B,S),(B,D)) (Vx)(x ∈ Γ1 - {B})

δ(q3.2,B,B)=(q3.2,(B,S),(B,I))

δ(q3.2,B,x)=(q3.3,(B,S),(B,I)) (Vx)(x ∈ Γ1 - {B})

δ(q3.3,B,x)=(q3.3,(B,S),(B,I)) (Vx)(x ∈ Γ1 - {B})

δ(q3.3,B,B)=(q4,(B,S),(B,S)) (Cinta 2 borrada vamos al punto 4)

1. Copia w de la cinta 1 a la cinta 2.

δ(q4 ,B,B)=(q4.1,(B,I),(B,S))

δ(q4.1 ,x,B)=(q4.2,(x,I),(x,I)) (Vx)(x ∈ Σ) (Copia la entrada en la cinta dos)

δ(q4.2 ,B,B)=(q02,(B,S),(B,D))

1. Simula M2 sobre la cinta 2. Si M2 para en qR -> M para en qR, si M1 para en qA -> M para en qA, si M2 loopea -> M loopea.

Para cada δ2(qi ,x)=(qj,y,m) se define δ(qi2,B,x)=(qj2,(B,S),(y,m))

(Simulación en cinta 2, nótese que si la simulación de M1 loopea entonces M loopeara)

δ(qA2,B,x)=(qA,(B,S),(x,S)) (Vx)(x ∈ Γ2)

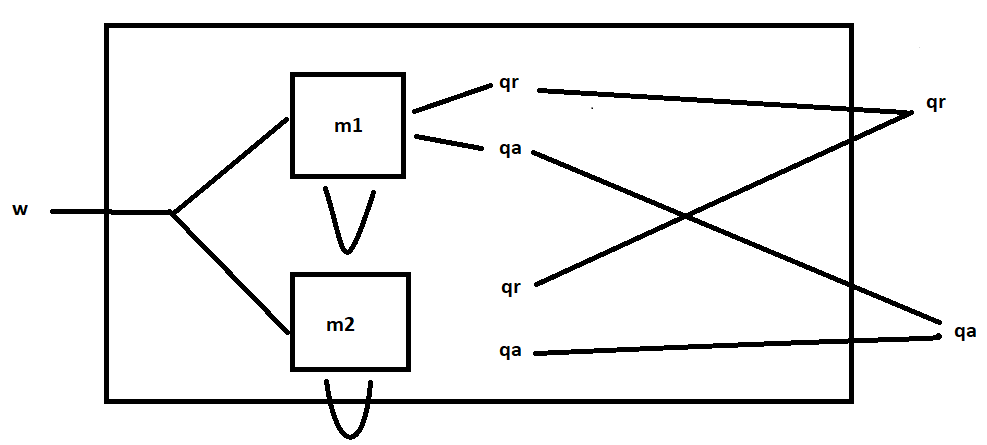
(Si M2 acepta, M también)

δ(qR2,B,x)=(q5,(B,S),(x,S)) (Vx)(x ∈ Γ2)

(Si M2 rechaza, M también)

Esta simulacion si puede usar B ya que luego no tendrá que borrar la cinta

ALERTA: Lo anterior no funcionaba, lo que sigue si:

Se va simulando un paso de cada uno

d) La unión de un número finito de lenguajes recursivamente enumerables es un

lenguaje recursivamente enumerable.

Demostraremos lo que dice el enunciado por inducción.

1. Caso base: tenemos 2 L, sean estas L1 y L2, ya en el inciso c demostramos que (L1 ∈ RE) AND (L2 ∈ RE) ⇒ L1 ∪ L2 ∈ RE. Por lo tanto, queda demostrado que para 2 lenguajes RE el resultado de la union también será RE.
2. Caso n+1:

Sean n lenguajes ∈ RE -> L1 ∪ L2 ∪ … ∪ Ln ∈ RE

(3) Definimos Lx=Ly ∪ Lw, Ly y Lw son RE, y por (1) Lx tambien lo sera.

Entonces podemos decir que:

L1 ∪ L2 ∪ L3 ∪… ∪ Ln = (3)

= Lx1 ∪ L3 ∪ … ∪ Ln = (3)

= Lx2 ∪ … ∪ Ln = (3)

= Lxi ∪ Ln = (3)

=Lxi+1

Gracias a la secuencia de pasos que realizamos sabemos que Lxi+1 ∈ RE

Y por lo tanto hemos demostrado por (1), (2) y (3) que La unión de un número finito de lenguajes recursivamente enumerables es un lenguaje recursivamente enumerable.

7) Para los casos a), b) y c) del punto anterior ¿valen las recíprocas? Justifique.

Las recíprocas serían estas:

a) L ∈ R ⇒ L ∈ R

Esto es cierto.

Sabemos que:

1. R = Co-R
2. L ∈ Co-R si y sólo si L ∈ R

Segun esto sabemos que si L ∈ R entonces L ∈ Co-R (1), si L ∈ Co-R entonces L ∈ R (2)

.: L ∈ R ⇒ L ∈ R

b) L1 ∩ L2 ∈ RE ⇒ (L1 ∈ RE) AND (L2 ∈ RE)

Se me ocurre que es falso ya que es una intersección, y podría ser que L1 y L2 no son RE debido a los elementos que no tienen en común, mientras que L1 ∩ L2 si es RE, pero tampoco se logró encontrar cuales serian estos elementos como para escribir el contra ejemplo.

Si tuvieramos que L1 ∈ RE y L2 ∈ RE

El ayudante contestó que está respondido en la explicación de los miércoles que le corresponde (14/10 o 21/10), la dejo para ver otro día.

c) L1 ∪ L2 ∈ RE ⇒ (L1 ∈ RE) AND (L2 ∈ RE)

El ayudante contestó que está respondido en la explicación de los miércoles que le corresponde (14/10 o 21/10), la dejo para ver otro día.

8) Si L es un subconjunto de un lenguaje recursivamente enumerable, ¿Puede afirmarse

entonces que L es recursivamente enumerable? Justifique.

L1 ∈ RE

L2 ⊂ L1

Buscando en internet encontré que:

Nótese que los lenguajes recursivamente enumerables no son cerrados con la diferencia ni el complementario.

* L ? P puede no ser recursivamente enumerable

De esto entiendo que los subconjuntos de un lenguaje RE no necesariamente son RE, pero no hallo lenguaje RE al que quitando elementos deje de ser RE.

El ayudante contestó que esto es correcto.

Dejo la búsqueda de contraejemplo para otro dia.

Aproveche las siguientes páginas para buscar la respuesta:

* <https://es.wikipedia.org/wiki/Lenguaje_recursivamente_enumerable>

9) Dado L1, un lenguaje recursivo cualquiera

L2 = {<M>| L(M) = L1}

L3 = {<M>| L(M) = L1 y M siempre se detiene}

Determine si (L2 – L3) = ∅ . Justifique su respuesta.

Primero voy a definir L2 y L3 de una forma mas coloquial:

* L2 es un lenguaje que acepta los códigos binarios de las MT que aceptan a L1.
* L3 es un lenguaje que acepta los códigos binarios de las MT que aceptan a L1 y siempre se detienen.

Sabemos que:

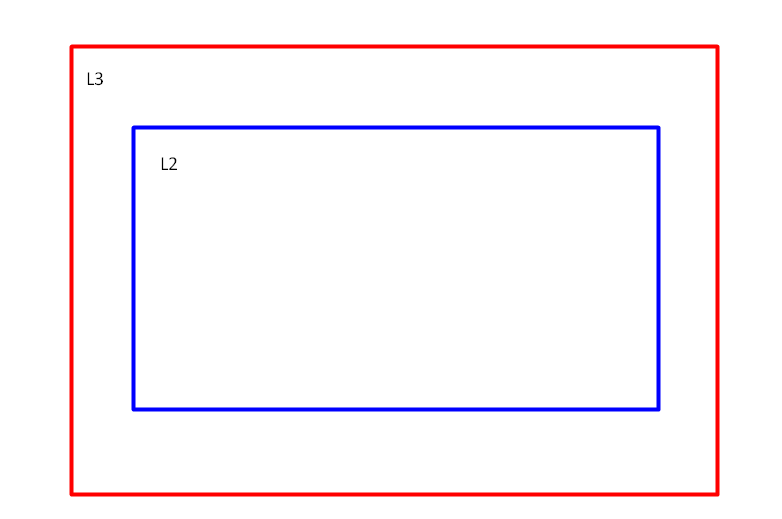
* L1 es R, por ende existe una MT M tal que L(M)=L1 y M siempre se detiene.

¿(L2 – L3) = ∅?

En comparación a los puntos anteriores este es particularmente sencillo de razonar:

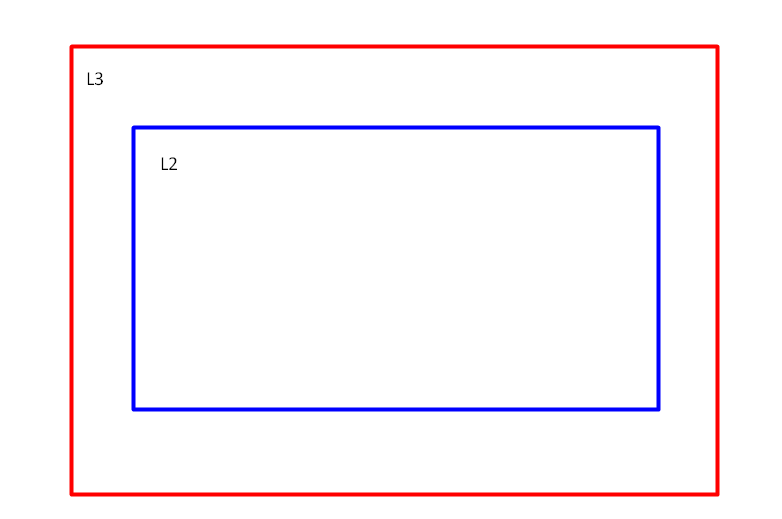
Sabemos que L1 es R por ende L1 es RE, osea que existen MT M tal que L(M)=L1 y M siempre se detiene y también existen MT M tal que L(M)=L1 y M no siempre siempre se detiene.

L3 solo se queda con las MT que si se detienen, mientras que L2 se queda con ambas, y es seguro que existirá un MT que no se detenga ya que L1 es RE, para visualizarlo de forma mas grafica un diagrama de Venn de esto sería así:



Quedo al reves el grafico.

Sabemos que en la diferencia de conjuntos A - B = C, el conjunto C mantendrá todos los elementos de A que no estén en B descartando los que estén en ambos o solo en B, entonces cuando hacemos L2 - L3 = LR nos quedaremos con la zona rayada en el siguiente gráfico:



Ahora diras “No hay una zona rallada”, exacto, por que no hay elemento de L2 que no pertenezca a L3 por ende LR = ∅.

.: Es cierto que (L2 – L3) = ∅.

10) Sean los lenguajes L = {<M>| M siempre se detiene} y LR = {<M>| L(M) ∈ R }.

Cuál es la afirmación correcta:

1. L ⊂ LR
2. L ⊃ LR
3. L = LR

Un lenguaje L es recursivo (R) o decidible si existe una MT M tal que L = L(M) y M siempre se detiene para todo input de Σ\*.

L es el conjunto de MT que siempre se detienen.

LR es el conjunto de MT que que aceptan lenguajes recursivos.

¿Sabemos que toda MT que se detiene acepta lenguajes recursivos y viceversa?

LR está incluido o es igual a L por que si la MT acepta un lenguaje recursivo por definición se tendrá que detener siempre.

L está incluido o es igual a LR por qué toda MT que se detiene siempre aceptará un lenguaje y ese lenguaje que sea aceptado por esta MT que sabemos que siempre se detiene necesariamente será R por definición de R.

.: Ya que sabemos que ambos lenguajes están incluidos o son iguales al otro diremos que los conjuntos son iguales.

.: La afirmación c es la correcta.

Aproveche las siguientes páginas para buscar la respuesta:

* <https://es.wikipedia.org/wiki/Lenguaje_recursivamente_enumerable>

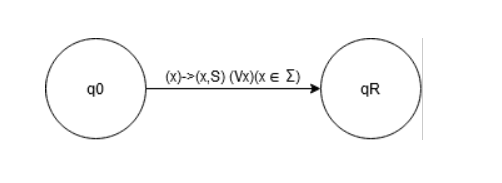
11) Encuentre una justificación para cada una de las siguientes afirmaciones

a) ∅ ∈ RE

Un lenguaje L es recursivamente enumerable (RE) si existe una MT M que lo acepte, es decir L=L(M).

En este caso L=∅, asi que demostrar que ∅ ∈ RE como armar una MT M tal que L(M)=∅ osea que todo input terminará en qR.

La describire gráficamente y con eso deberia bastar para mostrar que existe.



b) Si L es un lenguaje formado por una sola palabra, entonces L ∈ R

Armamos una MT tal que tenga tantos estados como símbolos tiene la palabra llamémosle w, de forma que de izquierda a derecha a cada símbolo le corresponde un estado y los estados se atravesaran en el mismo orden que los símbolos son leídos, cada estado se fija si el símbolo que está leyendo es igual al símbolo que le corresponde, si lo es mueve la cinta a la derecha y pasa al estado que sigue según el orden de los símbolos en la palabra, caso contrario (detecta cualquier otro símbolo que no sea el que nos interesa) va a qR, si atraviesa exitosamente todos los estados irá a qA.

De esta forma la MT siempre parara, para cada lenguaje formado por 1 palabra se la puede armar una MT como esta.

.: Si L es un lenguaje formado por una sola palabra, entonces L ∈ R

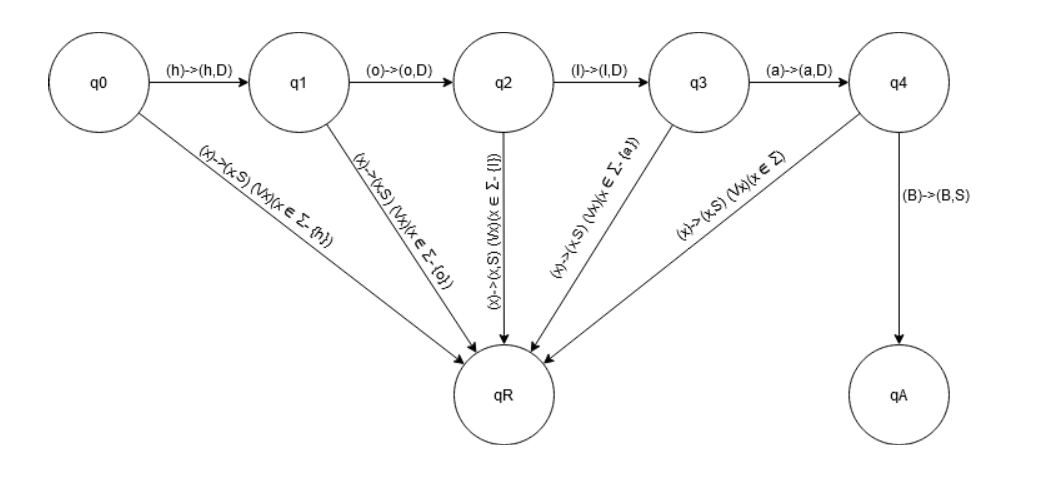
a continuación armo una MT M que acepta L={“Hola”}

¿Si tenémos una palabra/cadena w en particular, podríamos decir si hay una MT que la acepte? ¿cuál?.

Siempre se puede inventar una máquina que para todo símbolo "w" perteneciente a Sigma U {B} haga haga:

delta(q0,w)=(qA,w,S)

con lo cual aceptaría todas las cadenas incluso lambda.



c) Si L es un lenguaje finito, entonces L ∈ R

Investigando encontré que la union de 2 lenguajes recursivos siempre da por resultado otro lenguaje recursivo.

Sabemos por b que Si L1 es un lenguaje formado por una sola palabra, entonces L1 ∈ R

De esto podemos deducir que podríamos separar al lenguaje L que indica el enunciado en muchos lenguajes cada uno de estos que acepta una palabra del lenguaje y expresar L como la union de estos lenguajes.

.: Como la union de 2 o mas lenguajes recursivos da por resultado otro lenguaje recursivo entonces L que se puede expresar como la union de varios lenguajes recursivos también será recursivo.

Aproveche las siguientes páginas para buscar la respuesta:

* <https://es.wikipedia.org/wiki/Lenguaje_recursivo>

12) Demuestre que si el Halting Problem (HP) es un lenguaje recursivo entonces podría construirse una máquina de Turing que acepte el lenguaje universal Lu, y que se detenga para todo w ∈ Σ\* ¿Qué puede decir entonces sobre la recursividad de HP?

Lu = {(<M>, w) / M acepta w}

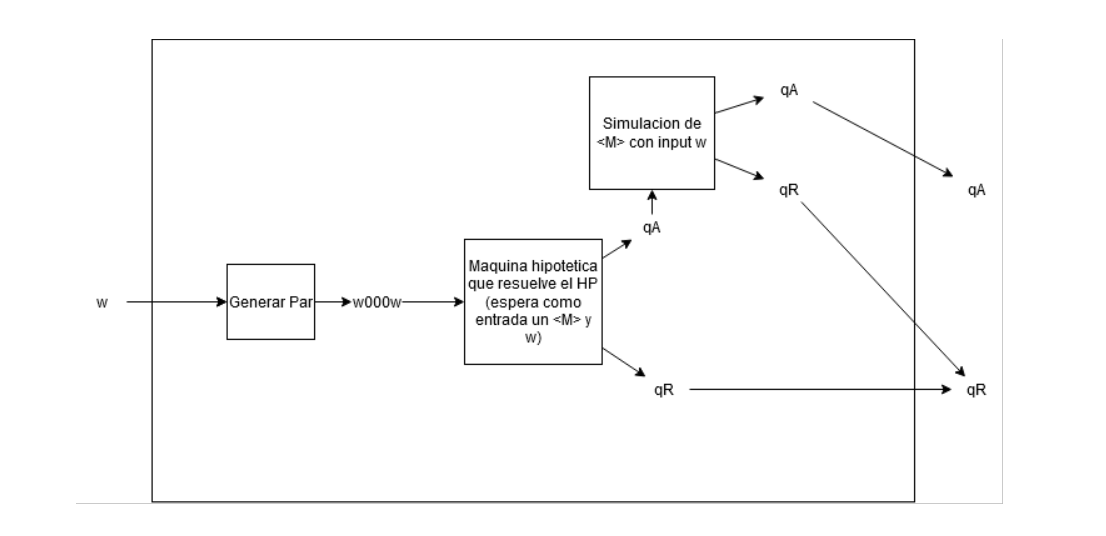
HP = {(<M>, w) / M se detiene con input w}

HP también conocido como el problema de la parada consiste en lo siguiente: dada una [Máquina de Turing](https://es.wikipedia.org/wiki/Máquina_de_Turing) M y una palabra w, determinar si M terminará en un número finito de pasos cuando es ejecutada usando w como dato de entrada.

Turing demostró que el HP es indecidible (no es recursivo).

Sabemos que Si HP fuera recursivo, la pregunta de si una MT dada se detiene con una entrada particular sería decidible. Ya que evidentemente existiria una MT que resuelve el HP y siempre se detiene.

Vamos a aprovechar esta hipotética MT para armar una MT que acepte el lenguaje universal Lu, y que se detenga para todo w ∈ Σ\*:



Esta máquina aprovecha la hipotética solución del HP para mostrarnos una MT que dado un par termina en qA si la máquina <M> termina en algún momento con entrada w, si no termina va a qR por que esto significa que <M> rechaza a w loopeando, si termina simula <M> con w, si lo rechaza, También, si lo acepta, entonces (<M>,w) es un par valido asi que va a qA.

Para este ejercicio HP debe ser necesariamente recursivo, caso contrario si loopeara entonces toda la MT también podría loopear y entonces no terminaría.

Aproveche las siguientes páginas para buscar la respuesta:

* <http://lya.fciencias.unam.mx/fhq/Cursos/TC/2016-2/tc-mt-modelos-ho.pdf> ç
* <https://es.wikipedia.org/wiki/Problema_de_la_parada>

13) Demuestre que LNV ∈ RE

LNV={(<M>)/L(M) ≠ ∅}

Entiendo que el lenguaje LNV está compuesto por todos los <M> tal que L(M) tenga como mínimo 1 elemento.

Para demostrar que LNV ∈ RE entonces debo demostrar que existe una MT M tal que L(M)=LNV.

¿Cómo podemos armar a M?

Puedo hacer una maquina Mnv que para cada <M> que reciba, genere las palabras de la forma:

Género los pares (i,j) en orden de su suma i+j , y entre los de de igual suma en orden creciente de i

Por cada par (i,j) generado simuló j pasos de la MT <M> sobre wi (i-ésimo string de sigma\* en orden canónico). Si M acepta a wi en esos j pasos, entonces hago que la máquina Mnv termine en qA.

