PRÁCTICA 3

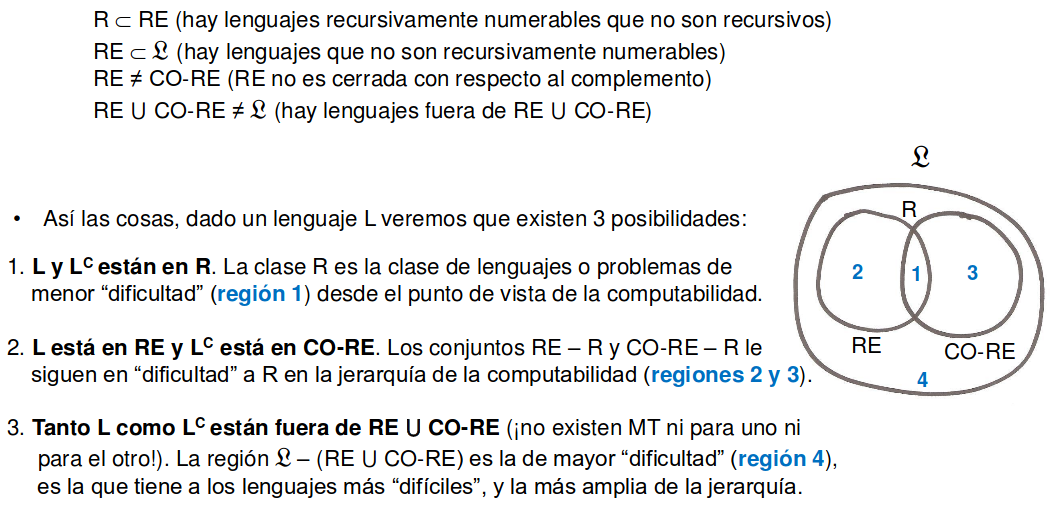
Jerarquía de la computabilidad. Problemas indecidibles

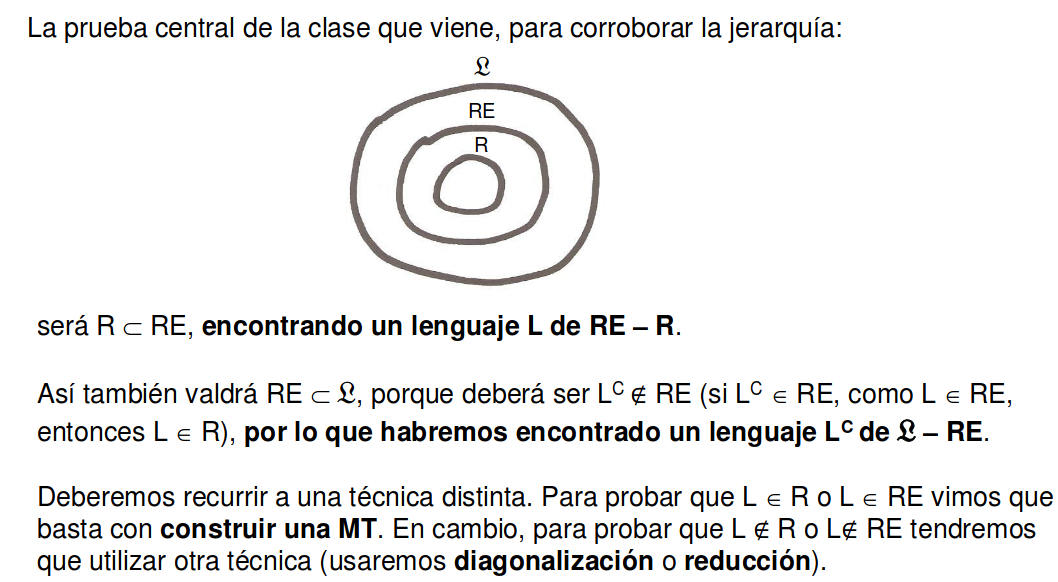
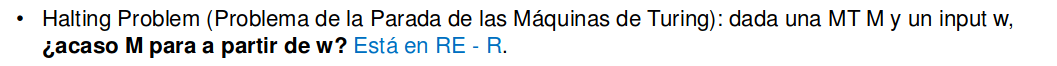
**Ejercicio 1. Recordar cómo probamos en la clase 3 que asumiendo R ⊆ RE se cumple RE ⊆ 𝔏.**

Dado un lenguaje L ∈ RE, si su complemento LC ∈ RE, L sería un lenguaje recursivo, es decir, valdría tanto que L ∈ R y LC ∈ R. Decimos entonces que LC ∉ RE

Entonces, al haber encontrado un lenguaje, LC el cual **no** pertenece a RE, sino que pertenece a CO-RE, queda demostrado que la clase RE está contenida en 𝔏; por lo tanto al ser 𝔏 más “grande”, quiere decir que hay lenguajes (es decir, problemas) que **no** están en RE, y por lo tanto no son computables

Otra forma de probarlo es por **diagonalización**. Al usar está técnica, se enumeran todos las MT que aceptan cada uno de los lenguajes tales que L ∈ RE; luego, se construye un lenguaje tal que ninguna de las MT puede aceptar (recordando que estamos probando todas las MT posibles que aceptan todos los lenguajes que pertenecen a RE); por lo tanto, se encuentra un lenguaje que ∉ RE y esto verifica efectivamente que RE ⊆ 𝔏

Otra solución:



**Ejercicio 2. Probar que los lenguajes LU = {(<M>,w) | M acepta w}, y HP = {(<M>,w) | M para sobre w} pertenecen a la clase RE.**

*Ayuda: las pruebas son similares a la desarrollada en la clase 3 para demostrar que D = { wi | Mi acepta wi } ∃ RE.*

**LU ∃ RE?**

Sea MTU la cual recibe un input w:

1. Rechaza si w no es un código de MT (<M>, w) válido
2. Genera M, y luego ejecuta M usando la codificación provista por w. Sólo si M acepta, MTU acepta; si M rechaza, MU rechaza
3. Si M loopea, MTU loopea

Cómo se puede construir una MT que acepta LU, pero puede loopear, queda demostrado que LU ∃ RE.

Prueba de la construcción:

1. Si w ∃ L(M), y M acepta al ejecutar w sobre ella, entonces MTU acepta
2. Si w ∃ L(M), y M rechaza al ejecutar w sobre ella, entonces MTU rechaza
3. si w ∄ L(M), MTU  rechaza
4. Si al ejecutar w sobre M, M loopea, MTU loopeará

**HP ∃ RE?**

Sea MTHP la cual recibe un input w:

1. Rechaza si w no es un código de MT (<M>, w) válido
2. Genera M, y luego ejecuta M usando la codificación provista por w. Sólo si M se detiene, es decir, si M llega al estado qA o qR, MTHP acepta
3. Si M loopea, MTHP loopea

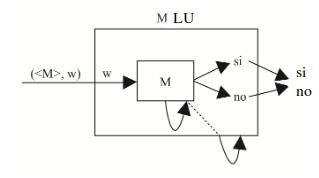
Cómo se puede construir una MT que acepta HP, pero puede loopear, queda demostrado que HP ∃ RE.

Prueba de la construcción:

1. Si w ∃ L(M), y M se detiene al ejecutar w sobre ella, entonces MTHP acepta
2. si w ∄ L(M), MTHP  rechaza
3. Si al ejecutar w sobre M, M loopea, MTHP loopeará

Otra solución:

LU = {(<M>,w) | M acepta w}. LU representa el problema de la pertenencia de una cadena a un lenguaje, o directamente el problema de la pertenencia.



La siguiente MT MLU acepta el lenguaje LU.

Dado un input w, MLU hace:

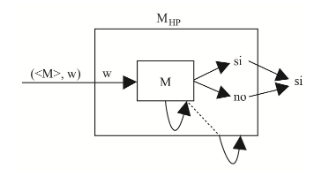
1. Genera <MLU> a partir del input

2. Ejecuta MLU sobre w , y acepta si y sólo si M acepta. Caso contrario rechaza.

Prueba:

Se cumple que L(M LU ) = LU: w ∈ L(M LU ) ↔ M LU acepta w ↔ w = (<M>, w) y M se

acepta a partir de w ↔ w ∈ LU.



La siguiente MT MHP acepta el lenguaje HP.

Dado un input w, MHP hace:

1. Genera <MHP> a partir del input

2. Ejecuta MLU sobre w , y se detiene si y sólo si Mhp se detiene.

Prueba:

Se cumple que L(M HP ) = HP: v ∈ L(M HP ) ↔ M HP acepta v ↔ v = (<M>, w) y M se

detiene a partir de w ↔ v ∈ HP.

**Ejercicio 3. En la clase 3 se probó que si HP ∈ R entonces R = RE, demostrando que sí existe una MT MHP que decide HP, entonces para cualquier lenguaje L de la clase RE existe una MT ML que lo decide. En realidad sólo se construyó ML. Se pide probar que efectivamente L(ML) = L.**

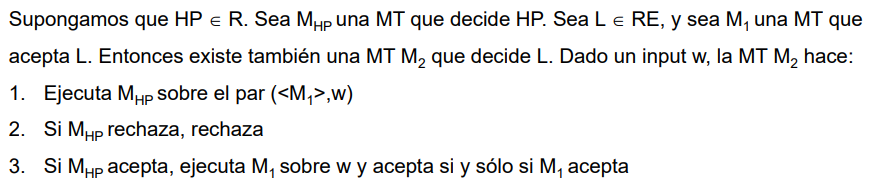
Si existe una MT que resuelve HP, y una MTL que acepta cierto lenguaje L ∃ RE, entonces uno podría construir, usando estas dos maquinas, una nueva MT que decida al lenguaje L.

La máquina construida verifica si MTL se detiene para un determinado input; si no se detiene, entonces rechaza

Si la máquina se detiene, entonces se puede ejecutar el input sobre MTL, y ésta podrá decidir si se acepta o rechaza. El resultado es la salida de la MT; siendo por hipótesis que MTL acepta el lenguaje, pero puede loopear, MTL efectivamente reconoce el lenguaje; y por lo tanto, la MT construida también acepta el lenguaje, dado que su salida es la salida de MTL

Que HP ε R entonces R = RE significa que si hubiera una MT que decide HP, entonces para cada lenguaje de RE habría también una MT que lo decide. En otras palabras, si se pudiera resolver completamente HP se podría resolver completamente cualquier problema de RE.

Otra solución:

****

Pruebas:

1. Prueba de que M L se detiene siempre.
   1. v ∈ L→ con entrada v (<M 1>, w) M L **se detiene** en qA luego de que acepte M HP y acepte M1
   2. v ∉ L→ con entrada v, M L **se** **detiene** en qR luego de que rechace M HP, o en el caso de que acepte M HP y rechace M1
2. Prueba de que L(M L) = L
   1. v ∈ L ↔ con entrada v = (<M 1>,w) M L se detiene en qA ↔ con entrada v MHP se detiene en qA y M 1se detiene en qA
   2. v ∉ L ↔ con entrada v = (<M 1>,w) M L se detiene en qR ↔ con entrada v MHP se detiene en qA y M 1se detiene en qR o M HP se detiene en qR

**Ejercicio 4. Responder cada uno de los incisos (justificar).**

1. **¿Se puede decidir si una MT M, a partir de la cadena vacía λ, escribe alguna vez un símbolo no blanco?***Ayuda: ¿Cuantos pasos puede hacer M antes de entrar en loop?*

Si, se debe construir una MT que ejecute tantos pasos como sea necesario para que M pase por todos los estados.

1. Hacer i := 1
2. Ejecutar i pasos de M; si en algún paso M escribe un carácter no blanco, aceptar; si M se detiene sin escribir un carácter no blanco, rechazar
3. Si M pasó por todos sus estados, y no se ha detenido, quiere decir que entró en loop, entonces se debe rechazar
4. Sino, hacer i := i + 1 y volver a 2
5. **¿Se puede decidir si a partir de un input w, una MT M que sólo se mueve a la derecha para?** *Ayuda: ¿Cuantos pasos puede hacer M antes de entrar en loop?*No bastaría con verificar si la cantidad de movimientos a la derecha > |w|?
6. **¿Se puede decidir si dada una MT M existe un input w a partir del cual M para en a lo sumo 10 pasos?**  
   *Ayuda: ¿Hasta qué tamaño de cadenas hay que chequear?*Si, es muy similar a la MT del ejercicio 7. Hay que ejecutar a lo sumo 10 pasos considerando cadenas con tamaño < i, siendo i la cantidad de pasos que se están ejecutando.
7. **¿Se puede decidir si dada una MT M existe un input w de a lo sumo 10 símbolos a partir del cual M para?** *Ayuda: ¿En este caso se puede acotar la ejecución de M considerando la cantidad de pasos, la cantidad de celdas recorridas u otro parámetro?*

No se puede, con solo restringir el input no alcanza, la maquina , al no saber que hace, ni cual es la maxima cantidad de pasos posibles (tampoco se pueden acotar). Esto seria cómo resolver HP.

Otra solución:

**a. ¿Se puede decidir si una MT M, a partir de la cadena vacía λ, escribe alguna vez un** **símbolo no blanco? Ayuda: ¿Cuántos pasos puede hacer M antes de entrar en loop?**

Es fácil detectar cuando una máquina está loopeando al ver que ya ha pasado por todos los estados posibles. Para cada celda de la cadena tengo que tener en cuenta cada uno de los diferentes símbolos definidos, así como cada estado. De esta forma puedo hacer un cálculo para saber el número de pasos máximos que se podrían realizar sin que la máquina entre a loopear. Permite asegurar que luego de esa cantidad la máquina va a estar loopeando.

Define p como la cantidad de pasos máximos sin loopear como:

p = c \* |Q| \* |r|^c

Siendo C la longitud de la cadena.

|Q| el conjunto de estados de la máquina

|r| número de símbolos del alfabeto definido para la máquina.

Como se introduce una cadena vacía, C es 0 y por tanto p = 0

**b. ¿Se puede decidir si a partir de un input w, una MT M que sólo se mueve a la derecha para?**  **Ayuda: ¿Cuántos pasos puede hacer M antes de entrar en loop?**

De nuevo, siguiendo el cálculo antes mencionado, la maquina deberia parar antes de hacer

p = c \* |Q| \* |r|^c, siendo c la longitud (cantidad de símbolos que componen) de w. Si pasada esa cantidad de pasos la máquina sigue ejecutándose se puede asegurar que entró en loop.

**c. ¿Se puede decidir si dada una MT M existe un input w a partir del cual M para en a lo sumo 10 pasos? Ayuda: ¿Hasta qué tamaño de cadenas hay que chequear?**

Se debería chequear hasta cadenas de longitud 10. Es un problema decidible. Podríamos usar una MT Ms que ejecute la MT M primero con todas las cadenas de longitud 1, si la MT M acepta una de estas Ms acepta, caso contrario Ms ejecuta todas las cadenas con una longitud máxima de 2 repitiendo el mismo procedimiento. Si llegado el caso en que se prueban todas las cadenas con máximo de longitud 10 y la MT M no aceptó ninguna, entonces Ms rechaza.

**d. ¿Se puede decidir si dada una MT M existe un input w de a lo sumo 10 símbolos a partir del cual M para? Ayuda: ¿En este caso se puede acotar la ejecución de M considerando la cantidad de pasos, la cantidad de celdas recorridas u otro parámetro?**

No se puede acotar considerando la cantidad de pasos, ni la cantidad de celdas recorridas (no se sabe la longitud de w).

**Ejercicio 5. Explicar cómo enumeraría los números naturales pares, los números enteros, los números racionales y las cadenas de Ʃ\* siendo Ʃ = {0, 1}.**

**Números enteros**: 0, 1, -1, 2, -2, …, En otras palabras, el primer elemento es 0, luego n, -n, ...

**Números naturales pares:** Similar al anterior, el primer elemento es 0, luego 2n, -2n, …

**Números racionales:**

**Cadenas de Ʃ\* siendo Ʃ = {0, 1}**: Se debe llevar un contador n, en donde se escriben todas las permutaciones de Ʃ usando n símbolos. Otra forma de verlo podría ser “contando” en binario

n=1 → 0, 1

n=2 → 00, 01, 10, 11

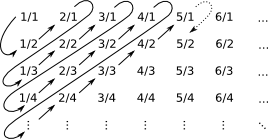
n=3 → 000, 001, 010, 011, 100, 101,111

Otra solución:

Estamos usando orden canónico.

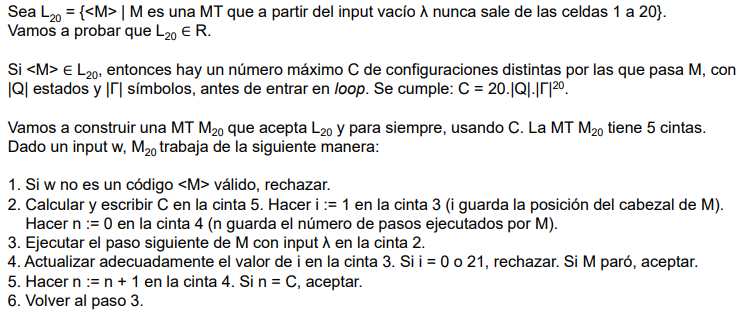
Lo haría de la siguiente forma:

* Naturales pares: 2, 4, 6, 8, 10, …
* Enteros: 0, 1, -1, 2, -2, 3, -3, …
* Cadenas de Ʃ\* siendo Ʃ = {0, 1}: 0, 1, 00, 01, 10, 11, 000, 001, ....
* Racionales: 1, ½, 2, ⅓, 3, ...



Consider the following infinite table, which clearly contains all of the postive rationals (with some duplicates upon considering reducing things to lowest form). Suppose we list the positive rationals in the following way. Starting with the upper left corner, snake back and forth along the diagonals as shown below, writing down the fractions as they appear on this path, **being careful to skip any (reducible) fractions that would have previously appeared in the list.**

**Ejercicio 6. Probar que la MT M20 construida en la clase 3 para decidir el lenguaje L20= {<M> | M es una MT que a partir del input vacío λ nunca sale de las celdas 1 a 20}, efectivamente para siempre y acepta dicho lenguaje.**

****

Pruebas:

1. Prueba de que ML se detiene siempre:
   1. w ∈ L 20→ con entrada w, M 20 se detiene en qA **acepta** luego de que pare M con con 0 < i < 21 (paso 4 del algoritmo), o en el caso de que n=C (paso 5).
   2. w ∉ L20 → con entrada w, M 20 se detiene en qR al rechazar por codigo invalido (paso 1 del algoritmo) o bien porque al simular M esta sale de las celdas permitidas, es decir que i = 0 o 21.
2. Prueba de que L(M 20) = L 20
   1. w ∈ L20 ↔ con entrada w M 20 se detiene en qA ↔ con entrada w = M se detiene con 0 > i > 21 o se llega a n = C

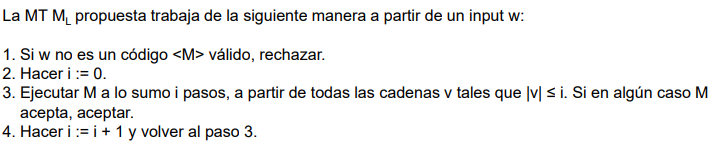
**Ejercicio 7. Probar que la MT ML construida en la clase 3 para aceptar L = {<M> | L(M) != ∅ } efectivamente cumple que L(ML) = L.**

MT ML básicamente funciona por contrareciproca: verifica si la MT M rechaza todos los inputs , ejecutando a lo sumo una determinada cantidad de pasos, para evitar loops. Además, las cadenas que se usan como input de la MT M se generan de forma tal que se prueban todas las posibles combinaciones, pero acotando siempre inputs de tamaño menor a la cantidad de pasos que se van a ejecutar; si esto no fuera así, la MT podría loopear en el primer input con i=0, 1, 2, 3 … n y nunca llegaría al segundo input

La MT ML propuesta trabaja de la siguiente manera a partir de un input w:

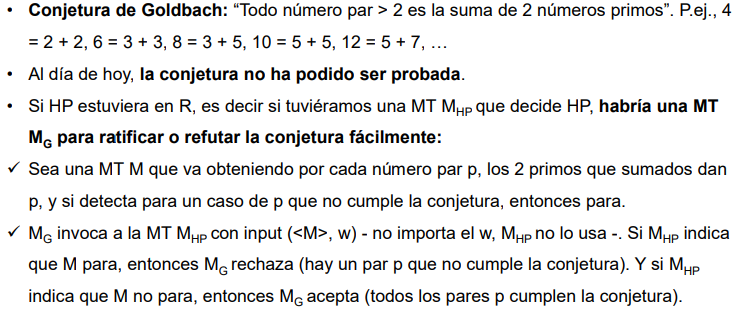
1. Si w no es un código <M> válido, rechazar.
2. Hacer i := 0.
3. Ejecutar M a lo sumo i pasos, a partir de todas las cadenas v tales que |v| ≤ i. Si en algún caso M acepta, aceptar.
4. Hacer i := i + 1 y volver al paso 3.

Otra solución:

****

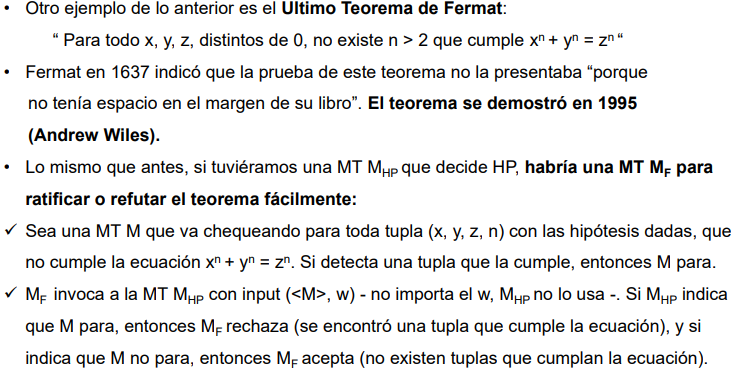
1. Prueba de que L(M L) = L
   1. w ∈ L ↔ con entrada w M L se detiene en qA ↔ con entrada w = M se detiene en qA
   2. w ∉ L, por como está definida, rechaza si el input es inválido o nunca para.

**Ejercicio 8. Probar que efectivamente las MT MG y MF presentadas en la clase 3, deciden la Conjetura de Goldbach y el Ultimo Teorema de Fermat, respectivamente, si se asume que HP es recursivo. En otras palabras, se pide verificar la correctitud de la construcción de ambas MT.**



Pruebas de MG (Conjetura de Goldbach):

1. Prueba de que MG se detiene siempre.
   1. v ∈ L→ con entrada v (<MHP>, w) MG **se detiene** en qA (acepta) si al invocar MT MHP acepta (dice que MT M no para).
   2. v ∉ L→ con entrada v, M L **se** **detiene** en qR (rechaza) luego de que rechace M HP (dice que MT M para, es decir que hay un p que no cumple)
2. Prueba de que L(MG) = L
   1. v ∈ L ↔ con entrada v = (<MHP>,w) MG se detiene en qA ↔ con entrada v MHP se detiene en qA

****

Pruebas de MF (Último Teorema de Fermat):

1. Prueba de que MG se detiene siempre.
   1. v ∈ L→ con entrada v (<M>, w) M F simula M HP con v → M HP rechaza v → M F **se detiene** en qA
   2. v ∉ L → con entrada v (<M>, w) M F simula M HP con v → M HP acepta v → M F **se detiene** en qR
2. Prueba de que L(MG) = L
   1. v ∈ L ↔ con entrada v = (<MHP>,w) MF se detiene en qR ↔ con entrada v MHP se detiene en qA

**Ejercicio 9. Una función f : A ⟶ B se dice que es total computable, si y sólo si existe una MT Mf que computa f para todo elemento a ε A. Sea la función fHP : Ʃ\* ⟶ {0, 1}, tal que:**

**f(x) = 1, si x = (<M>, w) y M para a partir de w**

**f(x) = 0, si x = (<M>, w) y M no para a partir de w, o bien x ≠ (<M>, w)**

**Probar que la función fHP no es total computable. Ayuda: Se podría probar que asumiendo que fHP es total computable, se llega a que HP es recursivo. En otras palabras, que se puede construir una MT que decide HP asumiendo que existe una MT que computa totalmente fHP.**

Dado fHP : Ʃ\* ⟶ {0, 1}, sabemos que existe una MT Mf que computa f para todo w ε Ʃ\*

Esta MT Mf, de acuerdo a lo definido en la función primero valida el input recibido y genera <Mw> (es decir <M> con el input w) para luego ejecutarla y aceptar si Mw acepta o rechazar si Mw rechaza.

Dado que no se sabe el lenguaje que acepta la MT M recibida, no se puede asegurar que la misma pare siempre (L podría no estar en R). Por lo cual al ejecutar la MT M con el input w en la MT Mf, si M entra en un loop, Mf también entrará en loop. Por ese motivo la MT Mf no podrá generar un resultado y por tanto no será computable totalmente.

**Ejercicio 10. Responder cada uno de los incisos (justificar).**

**a. Si L1 ∈ RE y L2 ∈ RE, ¿L1 – L2 ∈ RE?**

L1 - L2 representa a un L3 en donde solo se aceptan aquellas cadenas que L1 acepta y L2 rechaza.

Supongamos una MT M3 la cual simula la MT M1, la cual acepta L1 y la MT M2, que acepta L2. M3 acepta sólo cuando M1 acepta y M2 no lo hace. Es decir que la MT M3 acepta L1 - L2.

Supongamos ahora que queremos probar M3 con una cadena w. M1 acepta ante w y M2 no lo acepta.

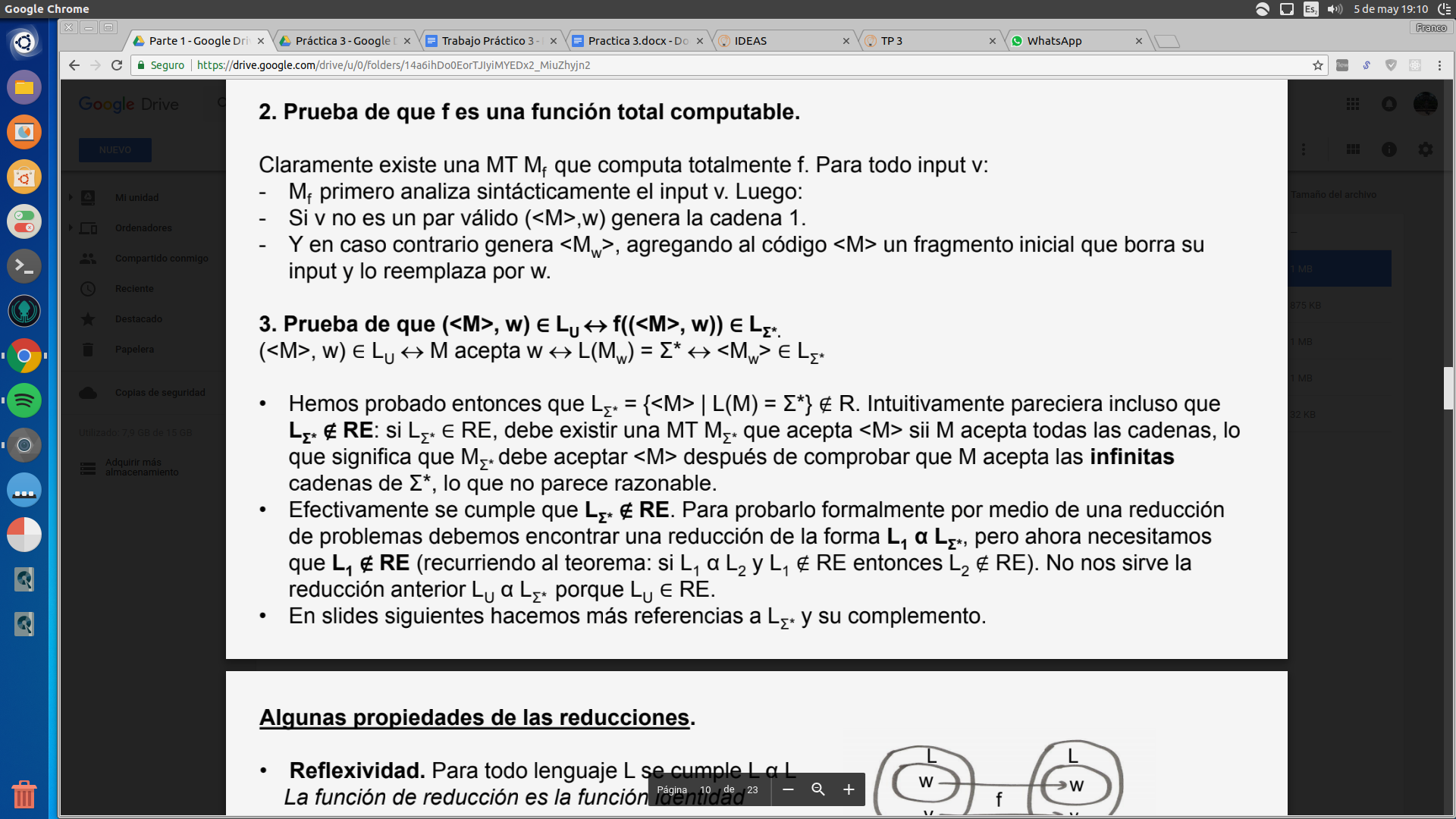
Al simular M1 y M2 en la MT M3, M1 entonces acepta sin problemas, pero la M2 se queda loopeando haciendo entonces loopear a M3, cuando esta debería haber aceptado.

Por lo tanto se podría decir que L1 - L2 no pertenece a RE.

**b. Si L1 ⋂ L2 ∈ RE, ¿L1 o L2 ∈ RE?**

L1 ⋂ L2 ∈ RE acepta sólo aquellas cadenas que son aceptadas individualmente tanto por L1 como por L2.

L sigma \*, y el complemento de éste, están afuera de RE. La intersección entre éstos es el conjunto vacío, que justamente sí está en RE (más específicamente R), pero los otros no. Por ende no puede asegurarse lo preguntado.

****

**c. Si L1 ⋃ L2 ∈ RE, ¿L1 o L2** **∈ RE?**

La unión de un lenguaje y su complemento da como resultado sigma estrella (todo). Considerando el caso de los lenguajes L sigma \* y L sigma \* complemento, se da lo mismo que arriba, ya que sigma estrella (la unión entre ambos) pertenece a R y RE, pero no los otros lenguajes.

**Ejercicio 11. Explicar (informal pero claramente) cómo sería una MT que genera la n-ésima fórmula booleana satisfactible, cuya sintaxis contiene variables de la forma xi, los operadores lógicos del conjunto {¬, ^ , v}, y paréntesis.**

La máquina debe ir generando diferentes cadenas a probar, colocándolas en una cinta de input. Con cada una de estas debería comprobarse que sea satisfacible. De ser así se debería incrementar un índice (el cual inicialmente debe tener el valor 0) que indique así que se encontró una nueva fórmula booleana que cumple.

Luego, se debe comprobar que el nuevo valor del índice coincide con el valor de n. Se darse el caso, la máquina puede parar de manera exitosa, enviando por una cinta de output la última cadena que cumplió. En el caso de que el índice sea distinto, se deberá continuar con el proceso generando nuevas cadenas.