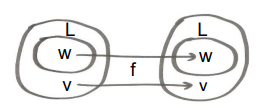
**Práctica 4**

**Reducciones de problemas. MT restringidas.**

**Ejercicio 1. Considerando la reducción de HP a LU descrita en clase, responder:**

**a. Explicar por qué la función identidad, es decir la función que a toda cadena le asigna la misma cadena, no es una reducción de HP a LU.**

La función de identidad hace lo siguiente:

****

Recordemos además las definiciones de:

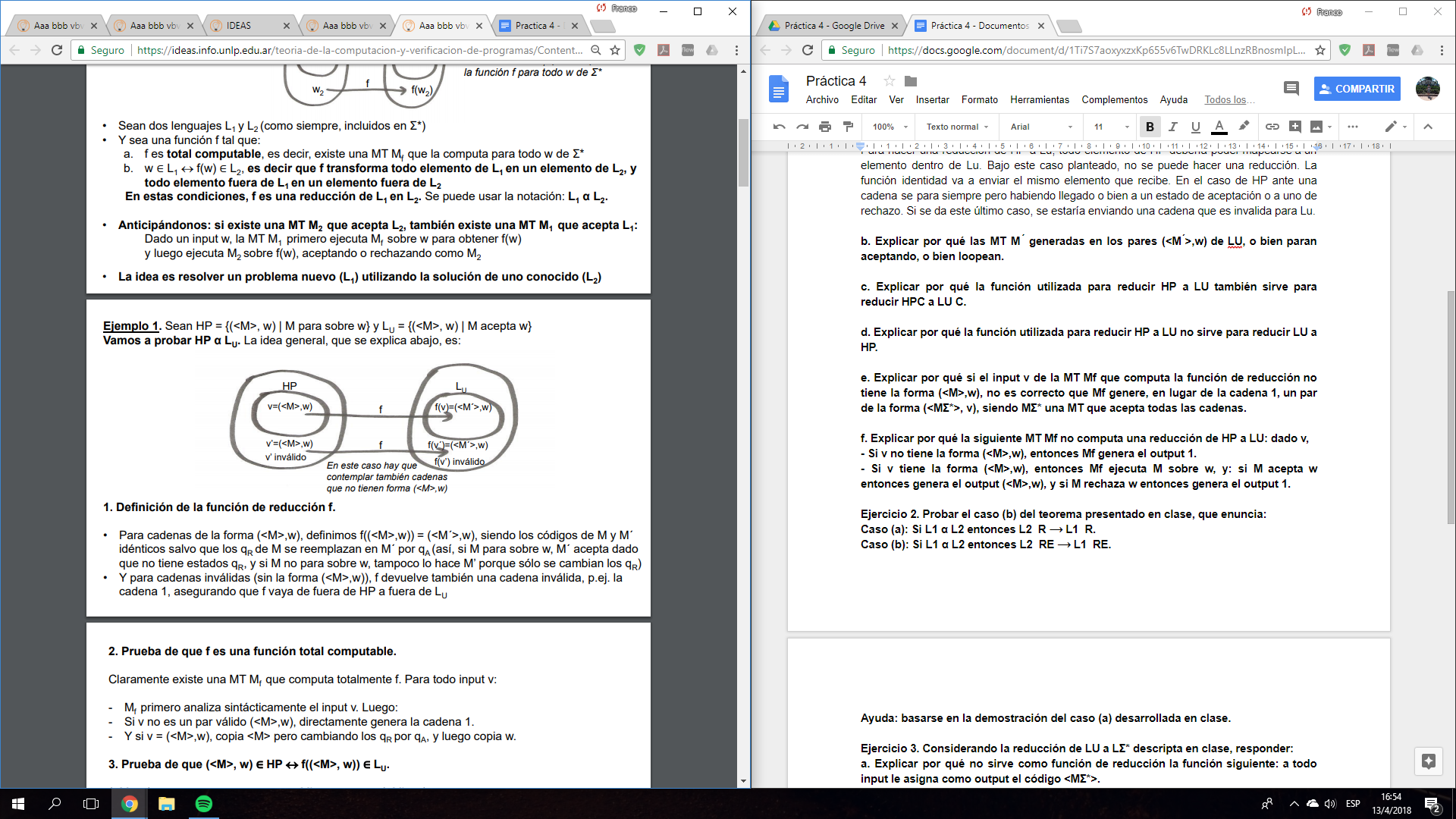
LU = {(<M>,w) | M acepta w}. **Es el problema universal de aceptación**.

HP = {(<M>,w) | M para sobre w}. **Es el problema de la parada (Halting Problem)**.

Para hacer una reducción de HP a Lu, todo elemento de HP debería poder mapearse a un elemento dentro de Lu. Bajo este caso planteado, no se puede hacer una reducción. La función identidad va a enviar el mismo elemento que recibe. En el caso de HP ante una cadena se para siempre pero habiendo llegado o bien a un estado de aceptación o a uno de rechazo. Si se da este último caso, se estaría enviando una cadena que es invalida para Lu.

Para este caso, entonces la función elegida no es totalmente computable.

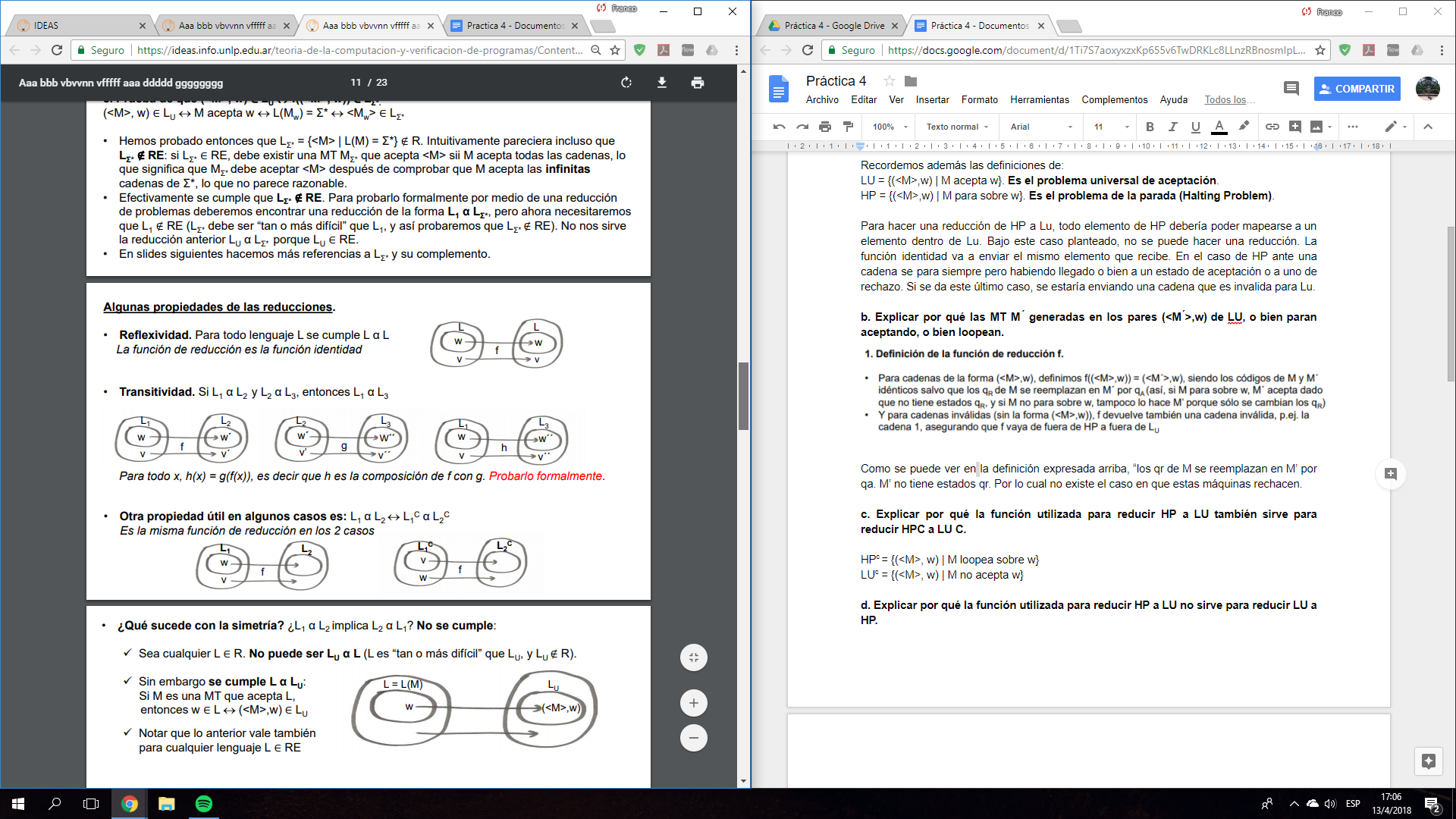
**b. Explicar por qué las MT M ́ generadas en los pares (<M ́>,w) de LU, o bien paran aceptando, o bien loopean.**

****

Como se puede ver en la definición expresada arriba, “los qr de M se reemplazan en M’ por qa. M’ no tiene estados qr. Por lo cual no existe el caso en que estas máquinas rechacen.

*Las MT M’ generadas en los pares (<M’>, w) de LU, paran aceptando porque dado (<M>, w) del lenguaje HP, siempre que la máquina para, sea cual sea su estado (qA o qR), la Mf al aplicar la función f(<M>,w) reemplaza todo estado qR en qA. Y si la Mf loopea, entonces la MT M’ también loopea.*

**c. Explicar por qué la función utilizada para reducir HP a LU también sirve para reducir HPC a LU C.**



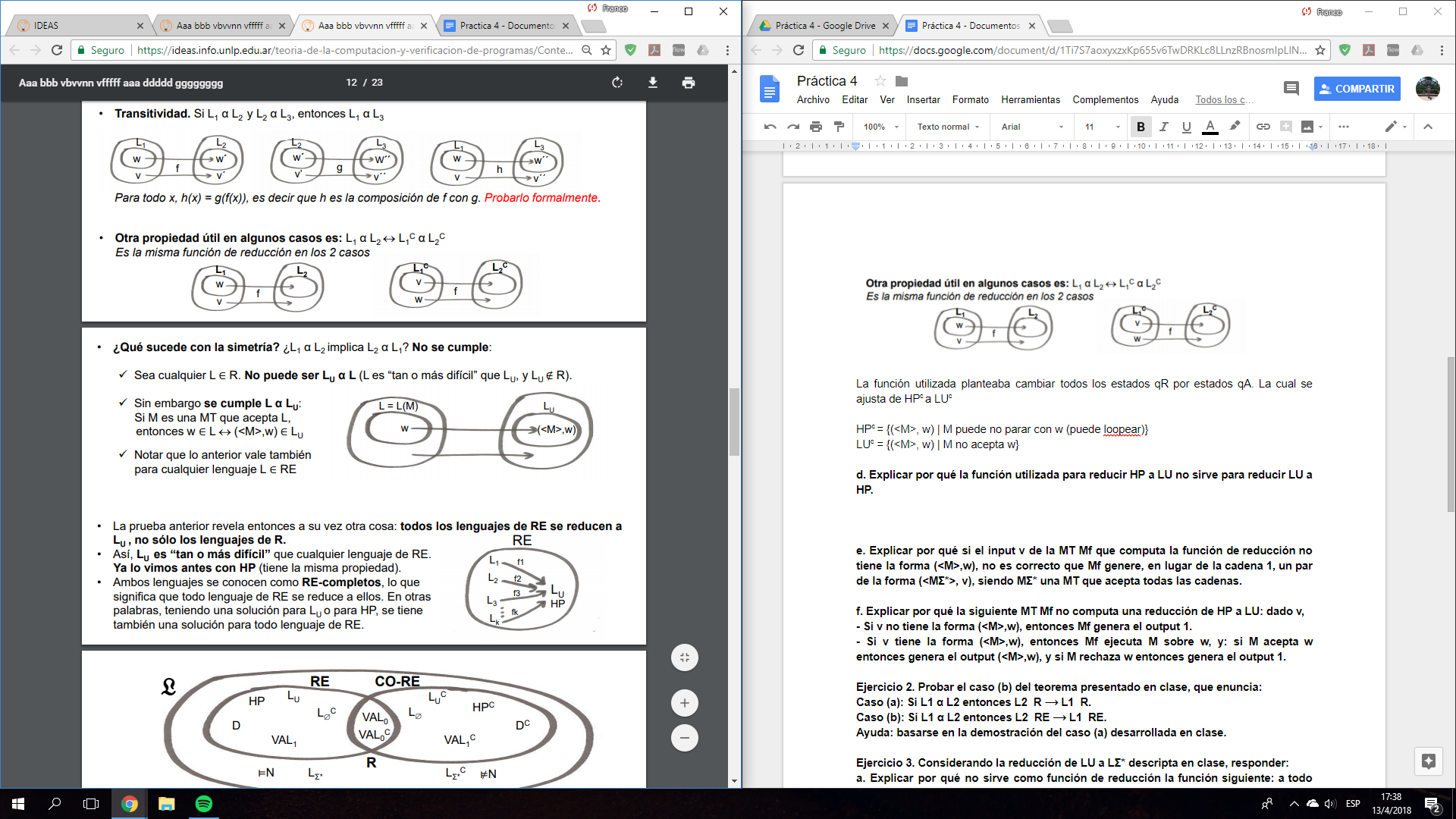
La función utilizada planteaba cambiar todos los estados qR por estados qA. La cual se ajusta de HPc a LUc

HPc = {(<M>, w) | M puede no parar con w (puede loopear)}

LUc = {(<M>, w) | M no acepta w}

#### ***La función utilizada para reducir HP a LU, sirve para reducir HPc a LUc , porque por definición, se cumple la siguiente propiedad:*** *Dado L1 y L2: si L1 α L2 → L1c α L2c.*

**d. Explicar por qué la función utilizada para reducir HP a LU no sirve para reducir LU a HP.**

****

Supongamos que aplicamos la misma función mencionada anteriormente. En este caso queremos reducir LU a HP. LU acepta siempre w si el mismo está en su alfabeto. En el caso de rechazar la función expresada mapearia ese estado de qR a un estado de aceptación o qA. Que HP luego aceptaría. Es decir que para algo que LU no acepta, HP termina aceptando, lo cual no es correcto.

*Podemos decir que la reducción no funciona porque dado un string que no pertenece a Lu puede generar un string que pertenece a HP. O dicho de otra manera, dado un string que pertenece a Lu puede generar un string que no pertenece a HP. (Y en ese caso, voy “de afuera a adentro”, lo que no es válido al hacer reducciones, sólo es válido ir “de adentro a adentro” y de “afuera a afuera”)*

*Ej: d*

*(Lu={(Mi, wk) | Mi acepta wk} y HP = {(M, w) | la MT M se detiene a partir de w)*

**e. Explicar por qué si el input v de la MT Mf que computa la función de reducción no tiene la forma (<M>,w), no es correcto que Mf genere, en lugar de la cadena 1, un par de la forma (<MƩ\*>, v), siendo MƩ\* una MT que acepta todas las cadenas.**

Ante un input inválido siempre se generaría una MT que siempre para en un estado qA. Esto puede resultar invalido porque se incumpliria con la definicion de funcion de reduccion para una reducción válida. Se estaría mapeando todo elemento de L1 (que puede estar dentro o fuera) a un elemento que esté dentro de L2.

*(<MƩ\*>, v) pertenece a Lu porque sea cual sea el input que ingrese, la máquina siempre lo aceptará porque MƩ\* acepta cualquier input, por lo tanto pertenece a Lu.*

*Si al aplicar la función de reducción luego de recibir un input inválido, ésta genera un par  (<MƩ\*>, v)* *(siendo MƩ\* una MT que acepta todas las cadenas), la otra máquina al recibir este input lo va a reconocer y aceptar, en vez de rechazarlo, que es lo que haría si recibe la cadena “1”. (voy de “afuera a adentro”, lo que no es válido en funciones de reducción)*

**f. Explicar por qué la siguiente MT Mf no computa una reducción de HP a LU: dado v,**

**- Si v no tiene la forma (<M>,w), entonces Mf genera el output 1.**

**- Si v tiene la forma (<M>,w), entonces Mf ejecuta M sobre w, y: si M acepta w entonces genera el output (<M>,w), y si M rechaza w entonces genera el output 1.**

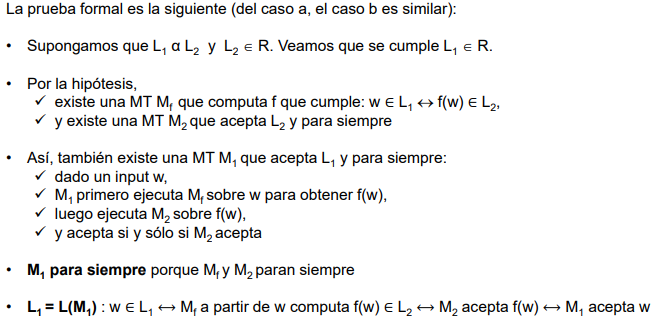
Al ejecutar el paso dos, si acepta, de lo único que nos aseguramos es que esa máquina de turing M paró ante el input w. Si M paró rechazando entonces estamos ante un caso en donde la función de reducción transforma un elemento dentro a HP a un elemento dentro de LU (cuando sabemos que ese elemento en realidad no lo esta).

**Ejercicio 2. Probar el caso (b) del teorema presentado en clase, que enuncia:**

**Caso (a): Si L1 α L2 entonces L2 R ⟶ L1 R.**

**Caso (b): Si L1 α L2 entonces L2 RE ⟶ L1 RE.**

**Ayuda: basarse en la demostración del caso (a) desarrollada en clase.**

****

**Caso b:**

Supongamos que L1 α L2 y L2 ∈ RE. Veamos que se cumple L1 ∈ RE.

• Por la hipótesis,

✓ existe una MT Mf que computa f que cumple: w ∈ L1 ↔️ f(w) ∈ L2 ,

✓ y existe una MT M2 que acepta L2

• Así, también existe una MT M1 que acepta L1:

✓ dado un input w,

✓ M1 primero ejecuta Mf sobre w para obtener f(w)

✓ luego ejecuta M 2 sobre f(w) y acepta si y sólo si M 2 acepta

• M1 puede llegar a loopear

• L1 = L(M1):

w ∈ L 1 ⟷ M f a partir de w computa f(w) ∈ L2 ⟷ M2 acepta f(w) ⟷ M1 acepta w

w ∉ L 1 ⟷ M f a partir de w computa f(w) ∉ L2 ⟷ M2 rechaza f(w) o loopea ⟷ M1 rechaza w o loopea.

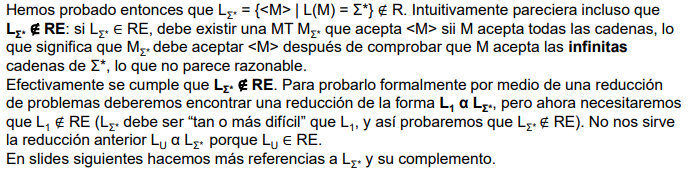
Adicionalmente es importante mencionar que ante un input inválido Mf rechaza.

**Ejercicio 3. Considerando la reducción de LU a LƩ\* descripta en clase, responder:**

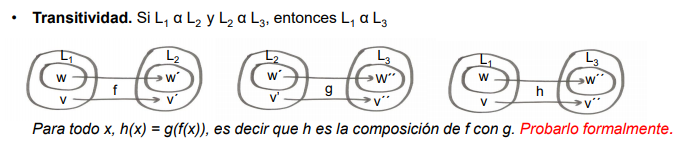
**a. Explicar por qué no sirve como función de reducción la función siguiente: a todo input le asigna como output el código <MƩ\*>.**

El problema ocurre ante los inputs que LU rechazaría. Si siempre se reemplaza por un código de la forma <MƩ\*> se plantea una máquina que acepta cualquier cadena. Es decir, estamos mapeando un elemento fuera de LU con uno dentro de LƩ\*. Lo mismo ocurre ante un input inválido.

**b. Explicar por qué la reducción descrita en clase no sirve para probar que LƩ\* ∉ RE.**

****

**Ejercicio 4. Probar formalmente que las funciones de reducción gozan de la propiedad transitiva. Ayuda: revisar la idea general comentada en clase; también basarse en la prueba que se haya desarrollado para el ejercicio 2, porque debería ser similar.**

****

Vamos a ver primero L1 a L2:

• Por la hipótesis sabemos que,

✓ existe una MT Mf que computa f que cumple: w ∈ L1 ↔️ f(w) ∈ L2 ,

✓ y existe una MT M2 que acepta L2

• Así, también existe una MT M1 que acepta L1:

✓ dado un input w,

✓ M1 primero ejecuta M f sobre w para obtener f(w) ( M f para siempre )

✓ luego ejecuta M 2 sobre f(w) y acepta si y sólo si M 2 acepta

• L1 = L(M1): w ∈ L 1 ⟷ M f a partir de w computa f(w) ∈ L2 ⟷ M2 acepta f(w) ⟷ M1 acepta w

Vamos a ver primero L2 a L3:

• Por la hipótesis sabemos que,

✓ existe una MT Mg que computa g que cumple: w ∈ L2 ↔️ g(w) ∈ L3 ,

✓ y existe una MT M3 que acepta L3

• Así, también existe una MT M2 que acepta L2:

✓ dado un input w,

✓ M2 primero ejecuta M g sobre w para obtener g(w)

✓ luego ejecuta M2 sobre g(w) y acepta si y sólo si M3 acepta

• L2 = L(M2): w ∈ L2 ⟷ M f a partir de w computa f(w) ∈ L3 ⟷ M3 acepta f(w) ⟷ M2 acepta w

**Ejercicio 5. Un autómata linealmente acotado (ALA) es una MT con una sola cinta con la restricción de que su cabezal sólo puede leer y escribir en las celdas en que se encuentra el input. Probar que el lenguaje aceptado por un ALA es recursivo. Ayuda: ¿en cuántos pasos se puede detectar que el ALA entra en loop?**

Es parecido a los ejercicios finales de la práctica anterior. Para calcular la cantidad de pasos máximos que la máquina puede realizar antes de entrar en loop se deben tener en cuenta una serie de datos como por ejemplo, la cantidad de estados existentes, la cantidad de símbolos o caracteres en el alfabeto así como también la longitud de la cadena w con la que se ejecuta la máquina.

p = n \* |Q| \* |Ʃ| ^ n

En particular se podría probar que el lenguaje que acepta la ALA es recursivo utilizando una máquina de Turing que simula dicha máquina. De esta forma si esta máquina acepta la cadena antes de los p pasos definidos entonces queda comprobado.

**Ejercicio 7. Construir un autómata finito que reconozca el lenguaje de las cadenas de {0, 1}\*, es decir todas las cadenas de 0 y 1 de cualquier tamaño incluso la vacía, tales que a todo cero le siga un uno. Ayuda: En general conviene primero construir el diagrama de transición de estados, porque da una idea de cómo construir el autómata finito.**

0

1

1

Sea el AF M = <Q, Ʃ, δ, q0, F> siguiente:

Q = { q0, q1 }

Ʃ = { 0, 1 }

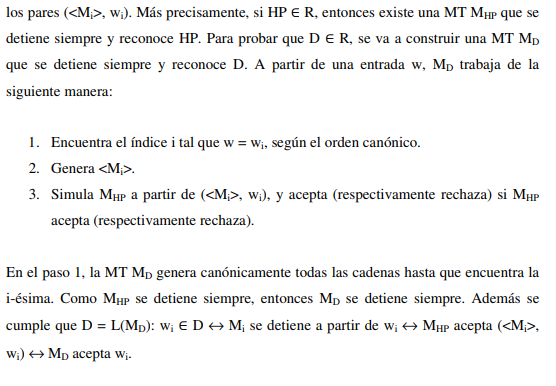
q0 = q0

F = {q0}

y la función δ es:

1. δ(q0, 1) = q0
2. δ(q0, 0) = q1
3. δ(q1, 1) = q0

**Ejercicio 7. Sea el lenguaje DHP = {wi | Mi para desde wi, según el orden canónico}. Encontrar una reducción de DHP a HP.**

****

**Ejercicio 8: Sea el lenguaje LØ = {<M> | L(M) = Ø }. Responder:**

**a. Encontrar una reducción de LUC a LØ. Ayuda: basarse en la idea de la reducción de LU a LƩ\*, es muy similar.**

1. Definición de la función de reducción f.

Para inputs válidos (<M>,w) se define:

f((<M>,w)) = <Mw>, donde Mw es una MT que:

(a) reemplaza su input por w,

(b) ejecuta M sobre w,

(c) acepta si y sólo si M acepta.

Notar que:

✓ si M rechaza w, entonces L(Mw ) = Ø porque Mw ignora sus inputs y los reemplaza por w (así que Mw acepta siempre), y por lo tanto está en LƩ\*

✓ si M acepta w, entonces L(Mw ) = Ʃ\* por la misma razón que antes (así que Mw rechaza siempre), y por lo tanto no está en LƩ\*

2. Prueba de que f es una función total computable.

Claramente existe una MT Mf que computa totalmente f. Para todo input v:

- Mf primero analiza sintácticamente el input v. Luego:

- Si v no es un par válido (<M>,w) genera la cadena 1.

- Y en caso contrario genera <Mw>, agregando al código un fragmento inicial que borra su input y lo reemplaza por w

3. Prueba de que (<M>, w) ∈ LUc ↔️ f((<M>, w)) ∈ LØ.

(<M>, w) ∈ LUc ↔️ M rechaza w ↔️ L(Mw ) = Ø ↔️ <Mw> ∈ LØ

**b. Considerando la reducción desarrollada en (a), ¿qué se puede decir de LØ, a qué clase de la jerarquía de la computabilidad pertenece?**

LUc α LØ

Como sabemos LUc ⊂ CO-RE (probado en la teoría), es decir que LUc ∉ RE

Por propiedad de reducción, LØ entonces tampoco ∉ RE, es decir que también pertenece a CO-RE.

L1 α L2

Si L1 no pertenece a RE entonces L2 no pertenece a RE

Si L2 pertenece a RE, entonces L1 pertenece a RE

LUc ∉ RE → LØ ∉ RE

**Ejercicio 9. Una fórmula booleana es satisfactible si una o más asignaciones de valores de verdad a sus variables la satisfacen (es decir, la hacen verdadera). Si todas las asignaciones la satisfacen la fórmula es una tautología (también se dice que es válida). Y si ninguna lo hace, la fórmula es una contradicción (también se dice que es insatisfactible).  
Por ejemplo, (x ⟶ y) ⟷ (¬y ⟶ ¬x) es una tautología.  
Sean VAL, SAT y UNSAT, respectivamente, los lenguajes de las fórmulas booleanas válidas, satisfactibles e insatisfactibles.**

**a) ¿Por qué la función identidad no es una reducción de VAL a SAT?**

VAL representa el lenguaje que acepta cuando ante una fórmula booleana recibida como input, con cualquier valor que se le asigne a sus variables da verdadero.

SAT acepta en cambio cuando al menos una de las posibles asignaciones da verdadero.

Si se usara la función identidad no se cumpliria que w ∈ VAL ⟷ f(w) ∈ SAT. Si tenemos una fórmula en la que por ejemplo solo una de las asignaciones da verdadero, VAL rechazaría cuando en SAT se debería aceptar. Se está mapeando un elemento que está fuera de VAL a un elemento que está fuera de SAT (cuando en realidad está dentro).

**b) Reducir VAL a UNSAT (en éste y en todos los ejercicios siguientes de reducciones de L 1 a L 2 , se debe describir la función de reducción f, probar que f es total computable, y probar que para todo w ∈ Ʃ\*: w ∈ L 1 ⟷ f(w) ∈ L 2).**

**Definición de la función de reducción f.**

Para inputs válidos (<M>, w) se define f(<M>, w) = <Mw>

Es importante mencionar que ante un input invalido, generar un output invalido igual a 1.

**Ejercicio 10. Probar mediante una reducción de problemas que L = {<M> | λ ∈ L(M)} ∉ R, siendo λ la cadena vacía. Ayuda: Basarse en alguno de los modelos de reducción vistos en clase.**

Para probar que el lenguaje L ∉ R es necesario encontrar otro lenguaje L1 tal que podamos reducir L1 a L. (Si se cumple la reducción, entonces L ∉ R)

1. Definición de la función de reducción f.

• Para inputs válidos (<M>,w) se define:

f((<M>,w)) = <M**λ**>, donde Mw es una MT que:

(a) reemplaza su input por **λ**,

(b) ejecuta M sobre **λ**,

(c) acepta sii M acepta.

• Para inputs sin la forma (<M>,w), se define como antes el output inválido 1

Claramente existe una MT Mf que computa totalmente f. Para todo input v:

- Mf primero analiza sintácticamente el input v. Luego:

- Si v no es un par válido (<M>,w) genera la cadena 1.

- Y en caso contrario genera <M**λ**>, agregando al código <M> un fragmento inicial que borra su input y lo reemplaza por **λ**.

3. Prueba de que (<M>, w) ∈ LU <-> f((<M>, w)) ∈ L.

(<M>, w) ∈ LU <-> M acepta **λ** <-> L(M**λ**) = L <-> <M**λ>** ∈ L

**Ejercicio 11. Probar mediante una reducción de problemas que L = {<M> | L(M) = S, con S ∈ RE y S ≠ ∅ } ∉ R. Ayuda: Basarse en alguno de los modelos de reducción vistos en clase.**

**Ejercicio 12. Construir una MT que genere todos los índices i tales que (<M i >, w i ) ∈ HP, según el orden lexical canónico.**