Práctica 4 FTC Las reducciones

Ejercicio 1. Considerando la reducción de HP a LU descripta 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.

Uno no puede reducir de HP hacia LU aplicando función de identidad por la correctitud:

- HP acepta cuando pare una maquina, LU acepta cuando acepta la maquina.
- La maquina que simula HP puede parar y rechazar pero la maquina HP aceptaría.
- Por los 2 puntos anteriores se debe modificar la maquina que le llega HP haciendo que todos los casos de parada sean aceptaciones, sino tendremos casos donde HP termina (para) y acepta y no esta en LU porque la maquina interna no acepta.
  Contra-ejemplo:
- La máquina Mx siempre rechaza dada una cadena w.
- MHP resuelve Mx, puesto que siempre para, la ejecuta 1 paso y la acepta.
- Si yo reduzco con identidad a LU, al ejecutar la maquina LU no acepta Mx, por tanto no se cumple la correctitud.

b. Explicar por qué las MT M2 generadas en los pares de salida (<M2>, w), o bien paran aceptando, o bien loopean.

La reducción descrita en clase modifica las qR de la maquina interna a que sean qA, esto evita que la maquina pare y rechazo, entonces el único caso de rechazo es que loopee.

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

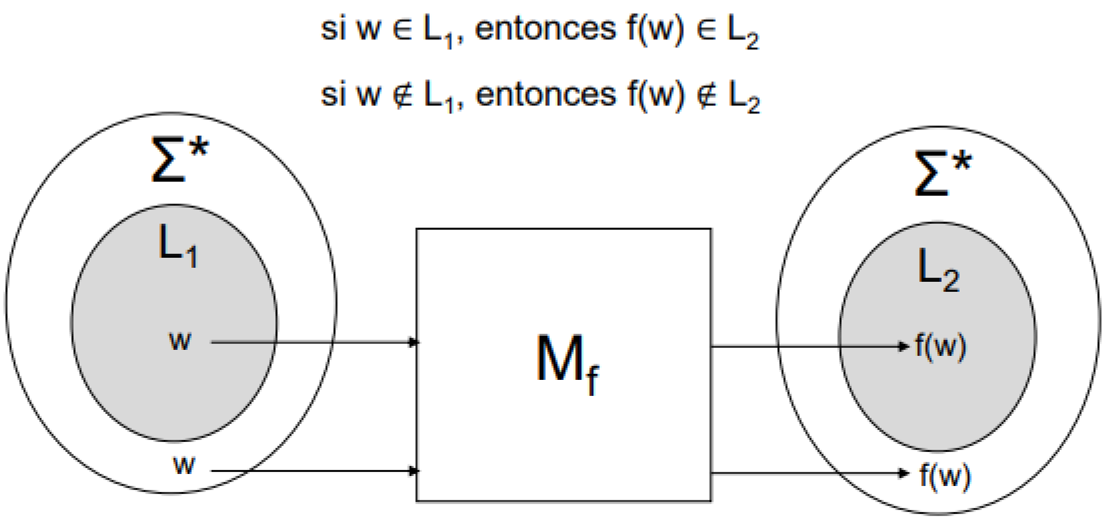
Propiedad: L1 <= L2 sii L1 C <= L2 C. La reducción es la misma.

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

Propiedad de simetría L1 <= L2 no implica L2 <= L1. Ahora porque en este caso no por que:

- Lo que esta en LU -> LHP, y lo que no esta en LU no debería terminar el HP.
- Si yo hago una maquina Mx que siempre falla Y para con una cadena w cualquiera y le aplico al reducción, esta terminara en HP porque efectivamente termina, pero nunca estuvo en LU para

arrancar.



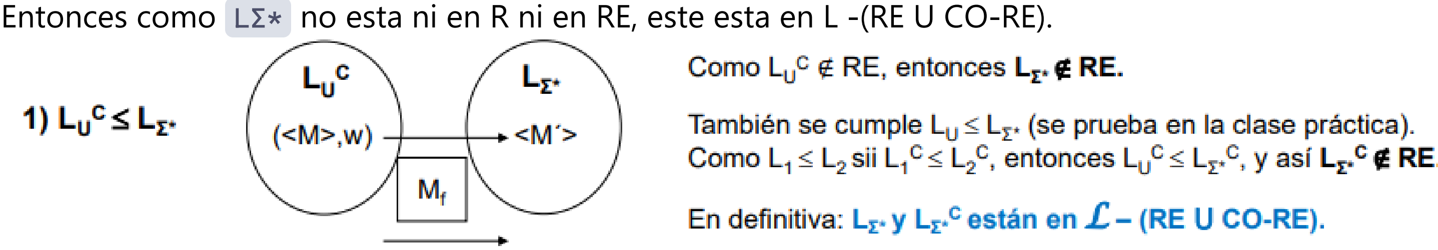
e. Explicar por qué la siguiente MT Mf no computa una reducción de HP a LU: dada una cadena válida (`<M>` , w), Mf ejecuta M sobre w, si M acepta entonces genera la salida (, w), y si M rechaza entonces genera la cadena 1.

Una reducción no puede ejecutar nada dentro suyo ya que puede quedar colgada, POR TANTO NO ES TOTAL COMPUTABLE.

Ejercicio 2. Sabiendo que LU ∊ RE y LU C ∊ CO-RE:

a. Probamos en clase que existe una reducción de LU a `LΣ*`. En base a esto, ¿qué se puede afirmar con respecto a la ubicación de `LΣ*` en la jerarquía de la computabilidad?

- Si L1 <= L2 y L1 ∉ R, entonces L2 ∉ R
- Como LU se puede reducir a `LΣ*` sabemos que `LΣ*` es igual o más complejo que LU y LU están en RE, nos dejaría con que `LΣ*` ∉ R.
- Además, como extra, LUC se puede reducir a `LΣ*`, como LUC ∉ RE, `LΣ*` ∉ RE.
- Entonces como `LΣ*` no esta ni en R ni en RE, este esta en L -(RE U CO-RE).



Ejemplo de reducción

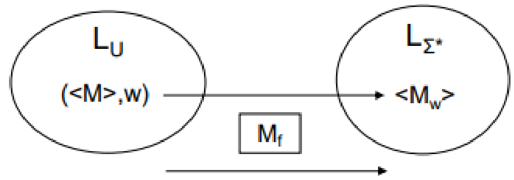
Sea el problema: dada una MT M, ¿acaso M acepta todas las cadenas de Σ^*?
El lenguaje que representa el problema, sabemos, es: $L_{\Sigma^*} = \{<M> \mid L(M) = \Sigma^*\}$. Probaremos con una reducción que $L_{\Sigma^*} \notin R$.
Ejercicio: Intuitivamente, ¿puede ser $L_{\Sigma^*} \in R$?. Más aún,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

Usaremos: **si $L_1 \leq L_2$ y $L_1 \notin R$, entonces $L_2 \notin R$.**
Así, hay que encontrar una reducción de la forma $L_1 \leq L_{\Sigma^*}$,
de modo tal que $L_1 \notin R$. Elegimos como L_1 el lenguaje L_U.

Reducción:

Definimos: $f((<M>,w)) = <M_w>$, tal que M_w es una MT que:

a) Reemplaza su entrada por w.
b) Ejecuta M sobre w.
c) Acepta sii M acepta.



Computabilidad:

Existe una MT M_f que computa f: genera $<M_w>$, agregando al código <M> un fragmento inicial que borra su entrada y la reemplaza por w.

Correctitud:

$(<M>, w) \in L_U \longrightarrow$ M acepta w $\longrightarrow M_w$ acepta todas sus entradas (¿por qué?) $\longrightarrow L(M_w) = \Sigma^* \longrightarrow <M_w> \in L_{\Sigma^*}$
$(<M>, w) \notin L_U \longrightarrow$ caso cadena válida: M rechaza w $\longrightarrow M_w$ rechaza todas sus entradas (¿por qué?) $\longrightarrow L(M_w) \neq \Sigma^* \longrightarrow <M_w> \notin L_{\Sigma^*}$
caso cadena no válida: M_w tampoco es una cadena válida $\longrightarrow L(M_w) \neq \Sigma^* \longrightarrow <M_w> \notin L_{\Sigma^*}$

25

b. Se prueba que existe una reducción de LU C a L🚫. En base a esto, ¿qué se puede afirmar con respecto a la ubicación de L🚫 en la jerarquía de la computabilidad?

Vacío no pertenece a RE porque LUC no pertenece a RE, como el complemento esta en L🚫C esta en RE (por construcción de una máquina), entonces L🚫 esta en CO-RE.

Ejercicio 3. Sea el lenguaje DHP = {wi | Mi para a partir de wi} (considerar el orden canónico}. Encontrar una reducción de DHP a HP. Comentario: hay que definir la función de reducción y probar su total computabilidad y correctitud.

- DHP es un lenguaje, wi que es resuelto con la Mi, que resuelve el Halting problem dado una maquina i para la cadena i.
- DHP <= HP en complejidad.
- DHP = {wi | Mi para a partir de wi} y HP = { (M, w) | M para con w}.
- Con todo lo anterior entonces la reducción debe transformar wi en un par (M, wi) donde M es Mi si wi es válido, sino generara una M que nunca se detiene por tanto no pertenece a HP.
- Computabilidad: Verifica que wi es una codificación valida para una MT Mi
  - Si wi es válida debemos extraer i de w, para este objetivo podemos generar todas las cadenas en orden canónico hasta llegar a wi contabilizando, y luego de forma similar haremos lo mismo generando el i-esimo código de maquina, construye el par (Mi, wi) correspondiente.

    - Si no es valida, extraeremos el i de la misma forma pero al llegar el momento de conseguir la maquina no habrá una Mi que se detenga, con ello se generara un par (Mloop, wi) donde Mloop es una MT que nunca termina.
    - Mf siempre termina, la generación, validación y concatenación son operaciones finitas.
- Correctitud:
    - wi ∈ DHP -> f(wi) = (Mi, wi) ∈ HP
    - wi ∉ DHP:
        - f(wi) = (Mloop, wi) y Mloop nunca para
        - f(wi) ∉ HP por virtud de nunca parar

Ejercicio 4. Sean TAUT y NOSAT los lenguajes de las fórmulas booleanas sin cuantificadores tautológicas (satisfactibles por todas las asignaciones de valores de verdad) e insatisfactible (insatisfactibles por todas las asignaciones de valores de verdad). Encontrar una reducción de TAUT a NOSAT. Comentario: hay que definir la función de reducción y probar su total computabilidad y correctitud.

- TAUT a NOSAT se puede hacer si negamos cualquier elemento de TAUT nosotros obtendremos una formula insatisfactible, viceversa debería funcionar, negar la formula (esencialmente envolver la fórmula en paréntesis y poner una ¬) es la reducción.
- Computabilidad:
    - Se toma la cadena original y se la envuelve en () y luego se pone un ¬. Esto es claramente computable en tiempo finito.
- Correctitud:
    - Si una cadena w ∈ TAUT, su negación f(w) ∈ NOSAT, es decir siempre es falsa.
    - Si w ∉ TAUT
        - Si no es una formula válida, entonces no es evaluable, lo que no la hace falsa ni verdadera, una Mt que acepte NOSAT la rechaza.
        - Si es una formula válida, al menos hay una posible solución donde no siempre es falsa.

Ejercicio 5. Se prueba que existe una reducción de LU C a LΣ *(y así, como LU C ∉ RE, entonces se cumple que LΣ ∉* RE). La reducción es la siguiente. Para toda w: f((`<M1>`, w)) = `<M2>`, tal que M2, a partir de su entrada v, ejecuta |v| pasos de M1 a partir de w, y acepta sii M1 no acepta.

Probar que la función definida es efectivamente una reducción de LU C a LΣ*. Comentario: hay que probar su total computabilidad y correctitud.

- La reducción dice: "Para toda w: f((`<M1>`, w)) = `<M2>`, tal que M2, a partir de su entrada v, ejecuta |v| pasos de M1 a partir de w, y acepta sii M1 no acepta."
- La reducción es valida, pero por que:
  - Lu={⟨M,w⟩I la MT M acepta la cadena w}, LuC es lo mismo pero la MT rechaza o loopea.
  - LΣ∗={⟨M⟩I L(M)=Σ∗} es el lenguaje de las MT que aceptan todas las cadenas de su alfabeto.
  - Entonces, una cadena de LuC que es válida rechaza o queda loopeando siempre, cuando es reducida a LΣ∗ por esta reducción se transforma es una maquina que ejecuta |v| pasos de M1 sobre w, y acepta sii M1 no acepta.
- Computabilidad:
  - Mf entonces recibe la tupla de entrada de (`<M1>`, w) y construye un código de maquina M2, programado para tomar la entrada arbitraria v como contador de pasos a ejecutar de M1 sobre la cadena w. Todo esto es computable en tiempo finito porque es copiar, escribir, etc.
- Correctitud:
  - Si `(<M1>, w)` ∈ LuC -> M2 ejecutara M1 sobre w una cantidad arbitraria de v pasos, en este contexto siempre falla por lo que aceptaría (condición si M1 no acepta), o se ejecuta v pasos y acepta por que termino el tiempo.
  - Si `(<M1>, w)` ∉ LuC -> M2 ejecutara M1 sobre w una cantidad arbitraria de pasos (v), entonces hay al menos un v donde M1 termina aceptando por tanto M2 rechaza, lo que lo hace estar fuera de LΣ*, dado que si hay un elemento del alfabeto que lo hace rechazar entonces falla.