Trabajo Práctico Nro 4 - Las Reducciones

Ejercicio 1

Considerando la reducción de HP a LU descripta en clase, responder:

1. 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 L_U.

Considerando la reducción de HP a L_U, la función identidad, es decir, *la función $f(v) = v$, no es una reducción de HP a L_U porque no cumple con la condición de correctitud para una reducción de lenguajes.*

Para que una función f sea una reducción del lenguaje L_1 al lenguaje L_2, debe satisfacer dos condiciones:

- *Condición de computabilidad:* Debe existir una Máquina de Turing (MT) M_f que compute f. La función identidad es computable por una MT que simplemente copia su entrada a la salida.
- *Condición de correctitud:* Para toda cadena w en el alfabeto Σ^*:
  - Si $w \in L_1$, entonces $f(w) \in L_2$.
  - Si $w \notin L_1$, entonces $f(w) \notin L_2$.

La Máquina de Turing M_{HP} puede parar por 2 casos:

- El código de máquina $< M >$ para, para la entrada w y termina en q_A, por lo tanto M_{HP} también termina en q_A.
- El código de máquina $< M >$ para, para la entrada w y termina en q_R, por lo tanto M_{HP} también termina en q_A.

En el *primer caso* cuando se realice la reducción va a ser correcto ya que al pasar el par $(< M >, w)$ por M_{HP} esta acepta y cuando ese mismo par se pasa por la función de identidad y se evalúa para L_U esto también es correcto ya que la máquina acepta.

El *segundo caso* es el que rompe la correctitud ya que al pasar el par $(< M >, w)$ por M_{HP} esta acepta pero cuando ese mismo par se pasa por la función de identidad y se evalúa para L_U supuestamente debería de ser correcto y aceptar pero como $< M >$ rechaza w, ese par en L_U no sería aceptado, ahí es donde la correctitud ya no se garantiza.

2. Explicar por qué las MT M_2 generadas en los pares de salida (<M_2>, w), o bien paran aceptando, o bien loopean.

Las MT M_2 generadas en los pares de salida (<M_2>, w) o bien paran aceptando, o bien loopean porque en la reducción descrita en la teoría se define a M_2 de la siguiente forma *"M_2 es como M_1, salvo que los estados q_R de M_1 se cambian en M_2 por estados q_A."* Esto se hace para poder garantizar la correctitud en la reducción, generando también así que las MT M_2 solo puedan aceptar o loopear.

3. Explicar por qué la función utilizada para reducir HP a L_U también sirve para reducir HP^C a L_U^C.

La función utilizada para reducir HP a L_U también sirve para reducir HP^C a L_U^C por una de las *propiedades de las reducciones* que nos dice que "$L_1 \leq L_2 \Leftrightarrow L_1^C \leq L_2^C$". La reducción es la *misma*.

4. Explicar por qué la función utilizada para reducir HP a L_U no sirve para reducir L_U a HP.

La función utilizada para reducir HP a L_U no sirve para reducir L_U a HP ya que en este caso *no se cumple la propiedad de Simetría de las reducciones*, esta propiedad nos dice que "$L_1 \leq L_2$ *no implica* $L_2 \leq L_1$", es más, esta propiedad solamente se puede cumplir en ciertos casos donde ambos lenguajes pertenezcan al conjunto R, condición que de base no tenemos.

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

La MT M_f no computa una reducción de HP a L_U ya que si tenemos un par $(< M >, w)$ donde $< M >$ no para con la entrada w la MT M_f se quedaría en loop, algo que no puede ocurrir ya que las *funciones de reducción* tienen que ser *total computables* y si esta loopea la propiedad no se cumpliría. Para estos casos donde sabemos que se pueden generar loops, las *funciones de reducción* deben de actuar como unas *"traductoras de sintaxis"* no deben computar ningún código ya que no garantizamos que la función pare, en casos donde trabajamos con *lenguajes decidibles* si podemos hacer que la *función de reducción* compute ya que al ser decidible sabemos que parará siempre ya sea por q_A o por q_R.

Ejercicio 2

Sabiendo que L_U ∈ RE y L_U^C ∈ CO-RE:

1. Probamos en clase que existe una reducción de L_U a $L_{\Sigma}*$. En base a esto, ¿Qué se puede afirmar con respecto a la ubicación de $L_{\Sigma}*$ en la jerarquía de la computabilidad?

Al saber que existe una reducción de L_U a $L_{\Sigma}*$ sabemos que $L_{\Sigma}*$ es igual o más complejo que L_U, como $L_U \in$ RE y $L_{\Sigma}*$ es igual o más complejo que L_U entonces deducimos que $L_{\Sigma}* \notin R$ ya que R es menos complejo que RE.

En segundo lugar, al saber que existe una reducción de L_U a $L_{\Sigma}*$ sabemos que existirá una reducción de L_U^C a ${L_{\Sigma}*}^C$ por propiedad de las reducciones, y como sabemos que $L_U^C \notin RE$ deducimos que ${L_{\Sigma}*}^C \notin RE$.

De forma parecida, al saber que existe una reducción de L_U^C a $L_{\Sigma}*$ y como $L_U^C \notin RE$ podemos asegurar que $L_{\Sigma}* \notin RE$.

Por definición también podemos ver que $L_{\Sigma}* \notin CO - RE$ ya que vimos que ${L_{\Sigma}*}^C \notin RE$ y para que $L_{\Sigma}* \in CO - RE$ su complemento debe pertenecer a RE.

*Como vemos que $L_Σ$ $\notin R$, que $L_{\Sigma}* \notin RE$ y que $L_{\Sigma}* \notin CO - RE$, en nivel de complejidad solo nos quedaría que $L_{\Sigma}* \in L$–$(RE \bigcup CO - RE)$.**

2. Se prueba que existe una reducción de L_U^C a $L_{\emptyset}$. En base a esto, ¿Qué se puede afirmar con respecto a la ubicación de $L_{\emptyset}$ en la jerarquía de la computabilidad?

Dado que existe una reducción de L_U^C a $L_{\emptyset}$ y sabiendo que $L_U^C \notin RE$ deducimos que $L_{\emptyset} \notin RE$. Por otro lado, se probó anteriormente que $L_{\emptyset}^C \in RE$, por lo tanto, siguiendo la definición podemos afirmar que *$L_{\emptyset} \in CO - RE$ ya que su complemento pertenece a RE.*

Ejercicio 3

Sea el lenguaje D_{HP} = {w_i | M_i para a partir de w_i} (considerar el orden canónico). Encontrar una reducción de D_{HP} a HP. *Comentario: hay que definir la función de reducción y probar su total computabilidad y correctitud.*

Idea General: Analizando los 2 lenguajes tenemos a D_{HP} = {w_i | M_i para a partir de w_i} y a HP = {$(< M >, w)$ | M se detiene a partir de w}. D_{HP} va a recibir una cadena w_i, al seguir el orden canónico, sabemos que esa cadena w_i va a tener una MT M_i que puede parar o no a partir de esa entrada, nosotros debemos pasarle a HP el par $(< M_i >), w_i)$. **Podemos plantear la reducción de esta forma:**

Reducción: Se define $f(w_i) = (< M_i >, w_i)$, tal que M_i es la MT asociada a la cadena w_i por orden canónico.

Computabilidad: Existe una MT M_f, que computa f: genera el par $(< M_i >, w_i)$, donde $< M_i >$ es la $i - ésima$ MT dada por el orden canónico. f es *total* ya que está definida para toda

entrada de su dominio ($\Sigma*$) y siempre encontrará la $i-ésima$ MT dada por el orden canónico, por lo tanto, no loopea, y es *computable* ya que la búsqueda de la $i-ésima$ MT y la creación del par $(< M_i >, w_i)$ se da en un tiempo finito.

Correctitud:

- $w_i \in D_{HP} \rightarrow M_i$ se detiene a partir de w $\rightarrow f(w_i) = (< M_i >, w_i) \in HP$
- $w_i \notin D_{HP} \rightarrow M_i$ no se detiene a partir de w $\rightarrow f(w_i) = (< M_i >, w_i) \notin HP$

Ejercicio 4

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

Idea General: Analizando los 2 lenguajes tenemos a $TAUT$ = {ϕ | ϕ es una fórmula tautológica (se cumple para toda asignación de verdad)} y a $NOSAT$ = {ϕ | ϕ es una fórmula insatisfactible (no se cumple para ninguna asignación)}. La clave es la lógica proposicional y usar una *negación* ya que $\text{Una fórmula } \phi \text{ es tautológica} \Leftrightarrow \text{su negación } \neg\phi \text{ es insatisfactible.}$ **Podemos plantear la reducción de esta forma:**

Reducción: Se define $f(\phi) = \neg\phi$, tal que $\neg\phi$ es la fórmula booleana insatisfactible asociada a la fórmula booleana tautológica ϕ.

Computabilidad: Existe una MT M_f, que computa f: genera la fórmula booleana insatisfactible $\neg\phi$, donde $\neg\phi$ es la misma fórmula ϕ que se recibe como entrada solo que se ve envuelta por una negación para hacerla insatisfactible. f es *total* ya que está definida para toda entrada de su dominio y ya que para toda fórmula booleana sin cuantificadores ϕ, su negación también es una fórmula válida, por lo tanto, no loopea. Y es *computable*, ya que la construcción sintáctica de $\neg\phi$ donde a la misma fórmula que se recibe de entrada se le agrega un símbolo de negación que la niega completamente, se da en un tiempo finito.

Correctitud:

- $\phi \in TAUT \rightarrow \phi$ es una fórmula booleana tautológica $\rightarrow f(\phi) = \neg\phi \in NOSAT$
- $\phi \notin TAUT \rightarrow \phi$ no es una fórmula booleana tautológica $\rightarrow f(\phi) = \neg\phi \notin NOSAT$

Ejercicio 5

Se prueba que existe una reducción de L_U^C a $L_{\Sigma*}$ (y así, como $L_U^C \notin$ RE, entonces se cumple que $L_{\Sigma*} \notin$ RE). La reducción es la siguiente. Para toda w: f((<M_1>, w)) = <M_2>, tal que M_2, a partir de su entrada v, ejecuta |v| pasos de M_1 a partir de w, y acepta sii M_1 no acepta. Probar que la función definida es efectivamente una reducción de L_U^C a $L_{\Sigma*}$. *Comentario: hay que probar su total computabilidad y correctitud.*

Total Computabilidad: Debemos mostrar que existe una Máquina de Turing (MT) M_f que, dada una entrada $(< M_1 >, w)$, siempre se detiene y produce la salida $< M_2 >$ tal como se define.

- Dada la codificación de una MT M_1 (representada por $< M_1 >$) y una cadena w, la MT M_f puede construir la codificación de una nueva MT M_2.
- La construcción de $< M_2 >$ implica especificar su comportamiento: al recibir una entrada v, M_2 debe primero calcular $|v|$, luego simular la ejecución de M_1 sobre w durante $|v|$ pasos. Finalmente, M_2 debe aceptar si y solo si M_1 no ha aceptado w en esos $|v|$ pasos.
- Cada uno de estos pasos puede ser realizado por una MT que siempre se detiene. Por lo tanto, M_f puede construir la descripción de M_2, $< M_2 >$, en un tiempo finito.
  La función f es total computable porque existe una MT M_f que puede calcular $f((< M_1 >, w))$ para cualquier entrada $(< M_1 >, w)$ y siempre se detiene.

Correctitud: Debemos demostrar que para toda cadena $(< M_1 >, w)$, se cumple que $(< M_1 >, w) \in L_U^C$ si y solo si $f((< M_1 >, w)) =< M_2 >\in L_{\Sigma*}$.

- *Caso 1:* Si $(< M_1 >, w) \in L_U^C$.
  - Esto significa que la MT M_1 no acepta la cadena w.
  - M_2 simula $|v|$ pasos de M_1 sobre w. Como M_1 no acepta w, no importa cuántos pasos simule M_2, M_1 nunca alcanzará un estado de aceptación en esos pasos.
  - Por lo tanto, para cualquier entrada v, M_2 siempre aceptará porque la condición "M1 no acepta" se cumple después de $|v|$ pasos (ya que M_1 nunca acepta w).
- *Caso 2:* Si $(< M_1 >, w) \notin L_U^C$.
  - Esto significa que la MT M_1 acepta la cadena w. Supongamos que M_1 acepta w en k pasos (donde k es un número finito de pasos).
  - Consideremos el comportamiento de M_2 sobre una entrada v cuya longitud $|v|$ es menor que k ($|v| < k$). Al ejecutar $|v|$ pasos de M_1 sobre w, M_1 aún no habrá aceptado. Por lo tanto, M_2 aceptará v.
  - Ahora, consideremos el comportamiento de M_2 sobre una entrada v' cuya longitud $|v'|$ es mayor o igual a k ($|v'| \geq k$). Al ejecutar $|v'|$ pasos de M_1 sobre w, M_1 habrá

alcanzado el estado de aceptación en algún paso $j \leq k \leq |v'|$. Por lo tanto, la condición "M1 no acepta" será falsa, y M_2 rechazará v'.