Indecibilidad. Reducciones (clases 3 y 4)

Sebastián Hernandez Corsi y Josefina Martinez Osti

Ejercicio 1. Probar que el lenguaje LU = {(<M>, w) | M acepta w} pertenece a la clase RE. Comentario: probarlo construyendo una MT.

Para probar que el lenguaje LU pertenece a RE, debemos demostrar que existe una MT_{LU} que lo acepta.

La MT que podemos construir es una que tome la codificación de una máquina <M> y una palabra w y simule la ejecución de w sobre M, lo cual podría resultar en q_a, q_r o en loop, debido a que el código de la máquina <M> recibida como entrada puede ser una máquina que loopee ante cualquier w dado.

MT_{LU} recibe las entradas y verifica que sean un par válido (<M>, w), rechazando si no lo son. En caso de que las entradas sean correctas copia <M> en otra cinta, lo que deja al cabezal de la cinta original posicionado en el principio de w. Una vez ahí, ejecuta <M> sobre la palabra w. si <M> acepta MT_{LU} acepta, si <M> rechaza MT_{LU} rechaza y si <M> loopea MT_{LU} también lo hace.

Ejercicio 2. Responder breve y claramente cada uno de los siguientes incisos (en todos los casos, las MT mencionadas tienen una sola cinta):

a. Probar que 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 un loop?

Se puede decir si M escribe algún símbolo blanco a partir de una cadena vacía ya que, el número de configuraciones de M es finito (tomando una configuración como el contenido de la cinta + el estado + la posición del cabezal). Podemos afirmar esto ya que se plantea que la entrada es la cadena vacía λ, lo que nos asegura que mientras que la máquina no escriba nada, el estado puede cambiar y el cabezal se puede mover, pero el contenido de la cinta no cambia. Teniendo esto en cuenta, podemos simular a M a partir de la cadena vacía y si repite alguna configuración sin haber escrito nada, entró en un ciclo y nunca va a escribir algo distinto de blanco. En este caso, al detectar que repitió la configuración habría que rechazar (no escribe nada). En el caso contrario, si en algún paso escribe un símbolo distinto de blanco, se acepta.

La cantidad máxima de pasos que puede hacer antes de loopear es |Q|

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

Teniendo una MT M que siempre se mueve a la derecha podemos decidir si para a partir de w o no dado que nos daremos cuenta si entra o no en loop si cuando termina de leer w se sigue ejecutando. Esto debido a que w es finita y la M solo se puede mover a la derecha, por lo que cuando termine de leer w solo habrán blancos y si no se detiene se van a repetir configuraciones.

La cantidad máxima de pasos que puede hacer antes de entrar en un loop es de |Q|+|w|.

c. Probar que se puede decidir si dada una MT M, existe una cadena w a partir de la cual M para en a lo sumo 10 pasos. Ayuda: ¿Hasta qué tamaño de cadenas hay que chequear?

Si, se puede decidir. Para hacerlo habría que tomar el alfabeto de M y generar cadenas de a lo sumo 10 caracteres. Luego hay que simular la ejecución de M sobre estas cadenas,

realizando a lo sumo 10 pasos para cada una. Si M se detiene antes de los 10 pasos para alguna de las cadenas, se acepta. En caso contrario se rechaza. Esto funciona ya que la cantidad de cadenas de longitud a lo sumo 10 es finita.

d. ¿Se puede decidir si dada una MT M, existe una cadena w de a lo sumo 10 símbolos a partir de la cual M para? Justificar la respuesta.
No, no se puede decidir porque no tenemos forma de darnos cuenta si comienza a loopear.

Ejercicio 3. Sea M1 una MT que genera un lenguaje recursivo L sin repetir cadenas y siguiendo el orden canónico. Probar que existe una MT M2 que decide L. Ayuda: cuidado con el caso en que L sea finito.
Lo que podemos hacer es simular a M1 (que imprime cadenas en orden canónico sin repetir) sobre M2. Necesitamos que M2 pueda recibir una w y decidir si dicha w pertenece a L. Para esto lo que podemos hacer es ir dejando que M1 imprima cadenas y, si la cadena c que imprime M1=w, M2 acepta. Si c>w entonces nos pasamos y la cadena no estaba, por lo que M2 rechaza, y mientras c<w se debe seguir simulando M1. Si M1 para (en el caso de que L sea finito) y todavía no se imprimió w, entonces M2 rechaza.

Ejercicio 4. 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.
Debido a que HP y LU no son el mismo lenguaje, puede darse que una cadena w $\in$ HP y esta misma cadena w $\notin$ LU llevando a que los elementos dentro de HP vayan fuera de los elementos de LU y también los casos contrarios.

b. Explicar por qué las MT M2 generadas en los pares de salida (<M2>, w), o bien paran aceptando, o bien loopean.
Esto debido al funcionamiento de la f(w) (función de reducción), ya que, los estados q_a son traducidos como estados q_a pero los estados q_r son intercambiados para la M2 como q_a dando que no exista ningún caso que pare rechazando, ya que cuando la M1 loopee la M2 también loopeara

c. Explicar por qué la función utilizada para reducir HP a LU también sirve para reducir HPC a LUC.
Esto debido a las propiedades de la reducción, ésta dice que si existe un lenguaje L1 con una reducción a L2 (L1 ≤ L2), también existirá una reducción entre sus complementos ($L1^c \leq L2^c$), por ende, por la reducción vista en clase del lenguaje HP a LU, sus complementos también tendrán la misma función de reducción.

d. Explicar por qué la función utilizada para reducir HP a LU no sirve para reducir LU a HP.
Las reducciones no son simétricas, es decir que si hay una reducción de un lenguaje L1 a otro lenguaje L2 (L1 ≤ L2) no implica exista si o si una reducción de L2 a L1 (L2 ≤ L1)

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 (<M>, w), y si M rechaza entonces genera la cadena 1.
En primer lugar, hay un problema con la Mf planteada ya que no se puede ejecutar M sobre w dado que HP no es decidible. Mf quiere ejecutar M sobre w antes de saber qué devolver y

esta M puede ser que entre en bucle, haciendo que Mf también lo haga. Esto haría que la función de reducción no sea total, lo cual es uno de los requisitos para ser efectivamente una función de reducción (debe ser computable total).
Por otro lado, los casos en que M rechaza generan la cadena “1”, la cual no pertenece a L_u, lo cual rompe la condición de equivalencia.

Ejercicio 5. Sabiendo que $L_U \in$ RE y $L_U^C \in$ CO-RE:
a. Probamos en clase que existe una reducción de L_U a LΣ*. En base a esto, ¿qué se puede afirmar con respecto a la ubicación de LΣ* en la jerarquía de la computabilidad?
Como hay una reducción de L_u a LΣ∗ ($L_U \leq$ LΣ∗) podemos afirmar que LΣ∗ se ubica en RE dentro de la jerarquía de clases de la computabilidad, ya que podemos resolver L_U usando un algoritmo que decida LΣ*. Esto implica que el lenguaje LΣ* con respecto al lenguaje L_U es tan difícil o más que L_U en la jerarquía de la computabilidad

b. Se prueba que existe una reducción de L_U^C a $L_\varnothing$. En base a esto, ¿qué se puede afirmar con respecto a la ubicación de L en la jerarquía de la computabilidad?
Esto quiere decir que el lenguaje $L_\varnothing \in$ CO-RE por propiedad de la reducción, esto quiere decir que el lenguaje $L_\varnothing$ con respecto al lenguaje L_U^c es tan o más difícil que este

Ejercicio 6. Sea el lenguaje DHP = {w_i | M_i para a partir de w_i} (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.
Debemos encontrar una función de reducción total computable. Podemos tomar $f(w_i)=(<M_i>,w_i)$, es decir una función que dada una entrada w_i de DHP(una cadena que representa la mt M_i con su entrada w_i) y simplemente “separar” la entrada en el par $(<M_i>,w_i)$.
Esto funciona ya que Si $w_i \in$ DHP, entonces $f(w_i) \in$ HP, ya que si $w_i \in$ DHP entonces la máquina M_i se para con w_i como entrada, y si ese es el caso, $(<M_i>,w_i) \in$ HP
Y si en cambio w_i no pertenece a DHP, $(<M_i>,w_i)$ tampoco pertenerecá a HP.

Ejercicio 7. Sean TAUT y NOSAT los lenguajes de las fórmulas booleanas sin cuantificadores tautológicas (satisfactibles por todas las asignaciones de valores de verdad) e insatisfactibles (ninguna asignación de valores de verdad las satisface), respectivamente.
Encontrar una reducción de TAUT a NOSAT. Comentario: hay que definir la función de reducción y probar su total computabilidad y correctitud.
Hay que encontrar una forma de tomar una fórmula tautológica y transformarla en una insatisfactible de manera que:

- Si una fórmula φ es tautológica (es decir, pertenece a TAUT), la fórmula transformada f(φ) será insatisfactible (es decir, pertenecerá a NOSAT).
- Si una fórmula φno es tautológica (es decir, no pertenece a TAUT), entonces f(φ)no será insatisfactible (no pertenecerá a NOSAT).

Lo que podemos hacer es simplemente negar φ: f(φ) = ¬φ, ya que si φ es tautológica entonces la ¬φ será insatisfacible y si φ no es tautológica entonces ¬φ no será satisfactible.
La función es computable ya que solo se debe invertir cada operador lógico de la fórmula (por ejemplo, si φ=p∧q, su negación ¬φ es ¬p ∨ ¬q)

Ejercicio 8. Se prueba que existe una reducción de $L_U{}^C$ a LΣ* (y así, como $L_U{}^C \notin RE$, entonces se cumple que LΣ* $\notin$ 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 M1 a partir de w, y acepta sii M_1 no acepta. Probar que la función definida es efectivamente una reducción de LUC a LΣ* (es total computable y correcta).

Nos están diciendo que una función de reducción de $L_U{}^C$ a LΣ* es: f((<M_1>, w))= <M_2>, donde M_2 es una mt que tiene entrada v y simula a M_1 sobre w durante |v| pasos para luego:
Si M_1 acepta w en ese tiempo, M_2 rechaza.
Si M_1 no acepta en $|v|$ pasos, entonces M_2 acepta.
Hay que ver que la función sea total computable:
La construcción de M_2 a partir de (<M_1>, w) es posible y efectiva, simplemente necesitamos una MT que simule a M_1 por $|v|$ pasos sobre w.
Luego hay que demostrar su correctitud:

- (<M_1>, w)$\in L_U{}^C$ es decir, M_1 no acepta w (rechaza o loopea). En este caso, cuando se ejecute M_2(v) va a simular M_1(w) por |v| pasos. Como M_1(w) no acepta nunca o rechaza entonces M_2(v) va a aceptar luego de |v| pasos, lo cual es correcto. Para todo v, M_2(v) acepta.
- (<M_1>, w)$\notin L_U{}^C$ es decir, M_1 acepta w luego de una cantidad x finita de pasos. Cuando M_2(v) simule M_1(w) por |v| pasos (con v>x), M_1 aceptará, y por lo tanto M_2(v) rechaza