TP 2

viernes, 2 de abril de 2021 19:51

4) a) No entendimos la pregunta

Ejercicio 1. Probar que el lenguaje L_U = {(<M>,w) | M acepta w} pertenece a la clase RE. *Ayuda: la prueba es similar a la desarrollada en la clase 3 para demostrar que D = { w_i | M_i acepta w_i} ∈ RE.*

Para probar que L_u ϵ RE nos basta con encontrar una máquina MT que acepte ese lenguaje.
La máquina puede ser una MT M_u tal que:
Si (<M>, w) no es un par válido o <M> no es un código de MT válido, para en qR.
Sino, simula M sobre w:
- Si M para en qA, M_u acepta.
- Si M para en qR, M_u rechaza.
- Si M loopea, M_u también va a loopear, por lo que rechazaría ese par.

Símbolos útiles:
⊦λϵ∉δΣΓ∀⊂⊆⇒ℒαΩΘ

Ejercicio 2. Explicar informal pero claramente cómo trabajaría una MT que genera la n-ésima fórmula booleana satisfactible (es decir que existe una asignación de valores de verdad que la hace verdadera), cuya sintaxis contiene variables de la forma x_i, los operadores lógicos del conjunto {¬, ∧, ∨} y paréntesis.

Suponiendo que nuestro input es n en notación unaria y con 3 cintas, la máquina debería hacer lo siguiente:
En la cinta 1 tenemos el input.
En la cinta 2 vamos a ir generando fórmulas booleanas.
En la cinta 3 vamos a comprobar que sean satisfactibles.

En la cinta dos vamos generando en forma ordenada fórmulas booleanas. Cuando se genera una fórmula, se copia ese string a la cinta 3, en donde se ejecuta una MT que reconozca si esa fórmula es satisfactible, en el caso que lo sea, la cinta 1 mueve su cabezal hacia la derecha. En el caso de que no sea satisfactible, la cinta 2 va a generar una nueva fórmula.

Cuando en la cinta 1 el cabezal apunte a blanco, la n-ésima fórmula booleana satisfactible se va a encontrar en la cinta 2. Si la cinta después de moverse no encuentra blanco, vuelve a realizar los pasos del párrafo anterior (generar fórmulas y ver si son satisfactibles).

Ejercicio 3. Justificar informal pero claramente cada uno de los incisos siguientes:
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?*
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?*
c. Se puede decidir, dada una MT M, si 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?*

4)a) hay que decir que la función de reducción "se rompe" porque va de adentro de HP a afuera de LU o de afuera de HP a adentro de LU

a) Si después de que la máquina haga C pasos, tal que C = |w| * |Q| * |Γ|, la máquina no escribió nada se podria decir que M entro en loop, ya que C es un número equivalente a la combinación de todos los símbolos que puede leer, escribir y estados en los que puede estar, por lo que se puede asumir que ya pasó por todas las funciones de transición.

b) Si, al igual que en el caso anterior, pero con C = |w| + |Q| se considera que entró en loop.
En el peor de los casos, la máquina se puede mover tantas veces como w (hasta llegar al blanco). Y una vez llegado a blanco, pueden haber estados que lean blanco, pero una vez que esos estados hayan sido recorridos la máquina ya no tiene más para hacer, por lo que podemos considerar que entra en loop.

c) Si, podemos usar la fórmula del a) pero despejar |w|, con C = 10, lo que nos va a decir la longitud del w para una cantidad de pasos menor o igual a 10.

|w| * |Q| * |Γ| <= 10
Despejando:
|w| <= (10 / |Q|) / |Γ|

Entonces, sabemos que hay que chequear hasta el tamaño de cadena menor o igual a (10 / |Q|) / |Γ| para que M se detenga a lo sumo en 10 pasos.

Ejercicio 4. Considerando la reducción de HP a L_U 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 L_U.
b. Explicar por qué las MT M´ generadas en los pares (<M´>,w) de L_U, o bien paran aceptando, o bien loopean.
c. Explicar por qué la función utilizada para reducir HP a L_U también sirve para reducir HP^c a L_U^c.
d. Explicar por qué la función utilizada para reducir HP a L_U no sirve para reducir L_U a HP.
e. Explicar por qué si el input v de la MT M_f que computa la función de reducción no tiene la forma (<M>,w), no es correcto que M_f genere, en lugar de la cadena 1, un par de la forma (<M_{Σ^*}>, v), siendo M_{Σ^*} una MT que acepta todas las cadenas.
f. Explicar por qué la siguiente MT M_f no computa una reducción de HP a L_U: dado v,
   - Si v no tiene la forma (<M>,w), entonces M_f genera el output 1.
   - Si v tiene la forma (<M>,w), entonces M_f 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.

a) porque no son el mismo lenguaje?

b) Esto se debe a que la función de reducibilidad cambia todos los estados "qR" de la M, y los pasa a "qA" en M', por lo tanto M' no tiene "qR" y no puede parar rechazando, solo parar aceptando o loopear.

c) Hay una propiedad que lo comprueba:
$L_1 \alpha L_2 \leftrightarrow L_1^C \alpha L_2^C$
(con la misma función de reducibilidad)
Como sabemos que la primer reducción funciona, sabemos que se cumple con los complementos.

También se puede pensar de manera lógica.
$HP^c = \{(<M>, w) \mid M \text{ no para sobre } w\}$ y
$L_u^c = \{(<M>, w) \mid M \text{ no acepta } w\}$

Si $w \in HP^c$ entonces M no para sobre w (ni en qA ni en qR), y por lo tanto M' (la que solo tiene cambiado qR por qA) tampoco va a parar sobre w, por lo que $w \in L_u^c$ también, ya que M' va a rechazar a w.
Si w no pertenece a HP^c es porque M paró en qA o qR, por lo que M' va a parar en qA, lo que hace que w tampoco pertenezca a L_u^c

d) Esta función no sirve para reducir L_u a HP, ya que la reducción puede fallar en el caso en donde la MT rechace w:
Si M rechaza w, entonces w no pertenece a L_u, y M' va a aceptar w, por lo que va a detenerse sobre w, lo cual hace que f(w) ∈ HP, y esto está en contra de la definición de la reducción, inciso b) de la teoría:
b. $w \in L_1 \leftrightarrow f(w) \in L_2$

e) Si el input v no tiene forma (<M>, w) significa que es un par inválido, por lo tanto no va a pertenecer a HP. Si nosotros con la función generamos el par ($<M_{\Sigma^*}>$, v), como esa MT siempre acepta, ese string va a pertenecer a L_u. Si así fuera, estaríamos teniendo una función de transición que está en contra de la definición (misma falta que en el inciso anterior).

f) Porque cuando Mf ejecuta M sobre w, esa M puede loopear, ya que nada nos asegura que se detenga, y entonces tendríamos una función de transición que nunca se detendría y no se cumpliría la reducción.

Ejercicio 5. Considerando la reducción de L_U a L_{Σ^*} 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 $<M_{\Sigma^*}>$.
b. Explicar por qué la reducción descripta en clase no sirve para probar que $L_{\Sigma^*} \notin RE$.

a) Porque puede darse el caso en donde w no pertenezca a L_u, y en ese caso, como la función de transición siempre genera la máquina $<M_{\Sigma^*}>$, ese f(w) va a pertenecer a L_{Σ^*}, por lo que estamos faltando a la propiedad de la reducción.

b) No sirve ya que la forma correcta de demostrar que L_{Σ^*} no pertenece a RE es haciendo una reducción desde un lenguaje que no pertenezca a RE. En este caso, la reducción se hace desde L_u, el cual es un lenguaje que si pertenece a RE, por lo que no nos sirve para demostrar eso.

Ejercicio 6. Probar formalmente que las funciones de reducción gozan de la propiedad transitiva. *Ayuda: revisar la idea general comentada en clase.*

Siendo L_1, L_2 y L_3 tres lenguajes tal que existe una reducción $L_1 \alpha L_2$ la cual es computada por una Mf_1 y una reducción $L_2 \alpha L_3$ computada por una función Mf_2, se podría hacer una reducción $L_1 \alpha L_3$ ejecutando estas funciones de transición de manera secuencial, es decir, Mf_3 (la máquina que reduce L_1 a L_3) conciste de pasar el input por la Mf_1, el output de esa Mf_1 va a ser el input de la Mf_2, y el output de la Mf_2 va a ser el output final de la Mf_3.

Ejercicio 7. Sea el lenguaje $D_{HP} = \{w_i \mid M_i \text{ para desde } w_i, \text{ según el orden canónico}\}$. Encontrar una reducción de D_{HP} a HP.

La idea principal es encontrar el índice i, para luego poder usar en HP la Mi y el wi, de esta manera ambos lenguajes se "comportarían" de manera similar.

Descripción de Mf:
Primero, necesitamos saber cual es el i de nuestro w_i. Para esto, vamos a generar en otra cinta los strings en orden canónico, llevando la cuenta en una tercer cinta, y por cada string generado lo comparamos con w_i, cuando el string generado sea igual, vemos el número que quedó en la tercera cinta, y así obtenemos i.

Luego, tenemos que generar la M_i, de la misma manera, vamos llevando un contador y creamos las máquinas en el orden canónico, hasta encontrar la M_i deseada.

Una vez tengamos la M_i, formateamos el output para que sea igual a ($<M_i>$, w_i)

Mf es total computable, ya que tanto encontrar el indice como generar la máquina M_i son acciones computables y que se pueden realizar para cualquier w dado.

$w_i \in D_{HP} \leftrightarrow f(w_i) \in HP$
Conociendo el funcionamiento de Mf, es lo mismo que plantear
$w_i \in D_{HP} \leftrightarrow <M_i, w_i> \in HP$
En ambos casos la máquina M_i se va a ejecutar sobre w_i, y por lo tanto se "comportan" de manera idéntica.

Ejercicio 8. Sean VAL y UNSAT los lenguajes de las fórmulas booleanas válidas e insatisfactibles (todas y ninguna asignación de valores de verdad las hace verdaderas, respectivamente). Encontrar una reducción de VAL a UNSAT.

Esta reducción es posible con una Mf que agregue un simbolo de negación al input recibido, esta es una máquina claramente total computable, ya lo que lo único que hace es agregar un símbolo (y, si se requiere, englobar entre "(" y ")" el string recibido antes de negarlo).
Mf va a cambiar por el opuesto todos los valores de verdad de las asignaciones que pueda tener esa función.
Por lo tanto, si w ∈ VAL, es porque todas sus asignaciones la hacen verdadera, por lo que f(w) sería una fórmula cuyas asignaciones siempre la hacen falsa, por lo que f(w) ∈ UNSAT
En caso de tener una fórmula que no pertenezca a VAL, invertirle los valores de verdad no va a generar una fórmula insatisfactible, por lo que esa w tampoco va a pertenecer a UNSAT. Sea w ∈ SAT (sin ser VAL) o w ∈ UNSAT, f(w) ∉ UNSAT.