

Probando LU ϵ RE se puede construir una máquina MU que acepte LU, tal que:

1. Si (<M>, w) no es un par válido parar en qR

2. En caso contrario separar <M> de w

3. Si <M> es un código inválido parar en qR

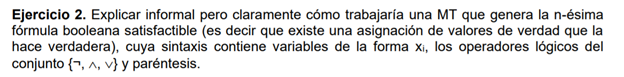
4. Simular M sobre w.

a. Si M para en qA => MU para en qA.

b. Si M para en qR => MU para en qR.

c. Si M loopea => MU también loopea.

entonces LU ϵ RE

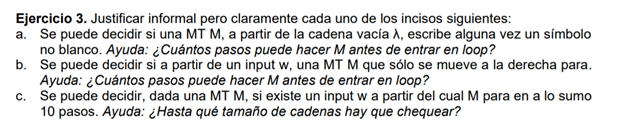
Σ = { 1 } // se utiliza de entrada un n unitario

Γ = { B, 1, A..Z, *¬, ^, v, (, )* }

La máquina constaría de 3 cintas, donde la primera sería el input, la cual comenzaría apuntando al primer 1 y cada vez que se encuentre una fórmula se movería a la derecha, hasta que se llegue a un blanco, cuando esto pasa, la máquina termina.

La 2da cinta es donde se irían creando los strings de forma ordenada canónicamente.

La 3ra cinta ejecutaría una MT que compruebe si el string generado es una fórmula booleana válida y satisfactible.

a) Si se podría decidir, ya que teniendo en cuenta que después de que M haga C pasos, tal que C es **C = |w| \* |Q| \* |Γ||w|** se podría decir que M entró en loop, ya que pasó por todos los estados posibles y no escribió nada.

b) Si, ya que después de C pasos, siendo C = |w| + |Q| se puede pensar que M loopeo.

En este caso la cantidad de estados se suman, porque a priori si la M solo se mueve a la derecha, el peor caso es que al llegar al primer blanco en q0, y luego todos los otros estados salvo qn cuando leen blanco pasan al estado siguiente, hasta que qn para.

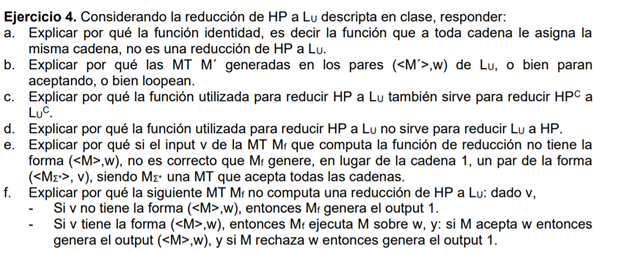
c) C = |w| \* |Q| \* |Γ|

Hay que intentar que C < 10

ð |w| \* |Q| \* |Γ| < 10

ð |w| < 10 / (|Q| \* |Γ|)

Si se puede decidir, puesto que para que pare en a lo sumo 10 pasos, el tamaño de la cadena debería ser menor a 10 / (|Q| \* |Γ|), de lo contrario no va a parar.

a. La función de reducción "se rompe" porque va desde adentro de HP hacia afuera de LU o desde afuera de HP hacia adentro de LU

b. Porque a la M original se le reemplazan los qr por qa, entonces, loopea o para en qa

c. La propiedad que lo comprueba es:



(con la misma función de reducibilidad)

Debido a que la primer reducción funciona, podemos saber que se cumple con los complementos.

Si lo pensamos de manera lógica:

HPc = {(, w) | M no para sobre w} y Luc = {(, w) | M no acepta w}

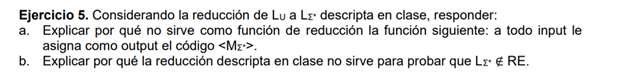
Si w ϵ HPc entonces M no para sobre w (ni en qA ni en qR), y por lo tanto M' (la cual únicamente tiene cambiado qR por qA) tampoco va a parar sobre w, por lo que w ϵ Luc también, porque M' va a rechazar a w. Si w no pertenece a HPc es debido a que M paró en qA o qR, por lo que M' va a parar en qA, por lo tanto w tampoco pertenece a Luc.

d. No sirve, ya que en el caso en donde la MT rechace w: Si M rechaza w, entonces w no pertenece a Lu, y M' va a aceptar w, por lo que va a detenerse sobre w, lo cual hace que f(w) ϵ HP, y esto está en contra de la definición de la reducción: 

e. Eso no lo puede hacer porque HP aceptaría, al recibir un w inválido tiene que generar un output que rechace siempre o bien sea una cadena inválida.

f. Porque al probar la máquina sobre el string puede loopear la Mf, y esta

tiene que ser total computable.

a. Porque la primer máquina puede rechazar y en ese caso la máquina output va a aceptar igualmente

b. Para demostrar que LƩ\* ∉ RE se necesita hacer una reducción desde un lenguaje L1 tal que L1 ∉ RE, en el caso de la teoría, se usa LU que sí pertenece a RE.



Sabiendo que L1 α L2 por medio de una Mf1 y que L2, α L3 por medio de una MF2. Se podría construir una MF3 que ejecute en serie las MF1 y MF2 pegando el output de la MF1 como input de la MF2 logrando la reducción de L1 a L3, siendo MF3 total computable.



La función de reducción f va a:

1. encontrar i, generando strings en orden canónico hasta llegar a wi
2. generar la maquinas en orden canónico hasta llegar a Mi
3. generar el output (<Mi>, wi) entonces

a. si <Mi> para sobre wi el par (<Mi>, wi) ϵ HP

b. si <Mi> no para sobre wi el par (<Mi>, wi) ∉ HP

siendo Mf total computable.



Para que la reducción sea posible utilizamos una Mf que añade un símbolo de negación al input recibido, como lo único que hace es agregar un símbolo, es una máquina total computable. Todos los valores de verdad de las asignaciones que tengan esa función se cambiarán por el opuesto.

Entonces si w *ϵ VAL* es porquetodas sus asignaciones la hacen verdadera, por lo que f(w) sería una fórmula a la cual sus asignaciones siempre la hacen falsa. En caso de que f(w) *ϵ UNSAT* tenga una fórmula que no pertenezca a VAL, no se podrá generar una formula insatisfactible invirtiendo sus valores de verdad, entonces esa w tampoco pertenecerá a UNSAT.

Sea *w ϵ SAT* (sin ser VAL) *o w ϵ UNSAT, f(w) ∉ UNSAT*