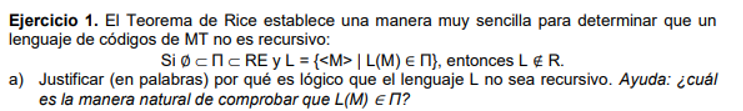
TP4



Necesitamos una MTD ML para poder comprobar si el lenguaje L es recursivo, que se encargue de reconocerlo en tiempo poly(n) y pare.

Esta MTD ML debería, para cualquier <M> que L reciba, comprobar que L(M) ϵ Π.

Para comprobar que L(M) ϵ Π, ML debería chequear si L(M) es igual a algún lenguaje de Π.

Para comprobar que L(M) es igual a algún lenguaje de Π, ML debería chequear L(M) con cada uno de los lenguajes de Π.

La comparación de L(M) con cualquier otro lenguaje (al que llamaremos L'), consiste en probar <M> sobre todos los strings de Σ\* y confirmar que aquellos lenguajes que <M> aceptó sean los mismos que los que acepta L'.

Vemos que simular la máquina <M> en todos los strings de todos los lenguajes no resuelve en tiempo poly(n) y para, entonces no se podría hacer una MTD que decide L.



Recurriendo al Teorema de Rice, planteamos el lenguaje L en los términos que se hace en el teorema.

Si ∅ ⊂ П ⊂ RE y L = { <M> | L(M) ∈ П}, entonces L ∉ R

con П = conjunto de lenguajes finitos

Con eso alcanzaría para comprobar que el L indicado en la consigna no pertenece a R.

Ahora tenemos que probar el conjunto de lenguajes finitos

Necesitamos probar que el conjunto de lenguaje finitos cumple con las propiedades de П (∅ ⊂ П ⊂ RE)

Demostraremos que ∅ ⊂ П, nos alcanza con señalar un lenguaje que se encuentre en el conjunto de lenguajes finitos, por ejemplo, con Σ = {1,2,3} un lenguaje podría ser L = {2}

Demostraremos que П ⊂ RE, como sabemos el problema de reconocer un lenguaje finito se encuentra en R, es decir, un lenguaje es recursivo porque se puede hacer una MT que pare siempre, ya que, para cualquier lenguaje finito, se podría hacer una MT con un estado por carácter y con eso se pueden reconocer todos los strings.

Ahora que demostramos que ∅ ⊂ П y П ⊂ RE, queda demostrado que ∅ ⊂ П ⊂ RE



Para probar que L pertenece a R debemos crear una MTD M' que lo decida.

La idea es construir una máquina que lea el input <M> y por cada estado escriba un 1 en la cinta 2.

Cuando termina de recorrer el input, se recorre la máquina 2, con un estado que determina si el input es par, y otro si es impar, alternando el estado cada vez que lee un 1 hasta encontrarse con un blanco y tener la respuesta sobre la paridad en el número de estados.

M' tiene 2 cintas. Con el input w en la cinta 1, M' hace:

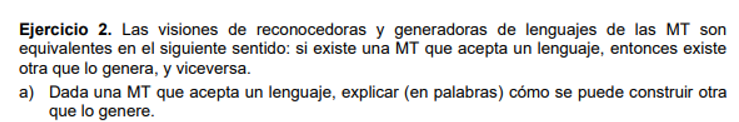
1. Lee w, y por cada estado leído (se reconoce fácilmente por la sintaxis de la <M>), escribe un 1 en la cinta 2.

2. Lee la cinta 2 en el estado qPar, si el estado qPar se encuentra con un blanco pasa a qA, si encuentra un 1 pasa a qImpar.

3. Si qImpar encuentra un blanco, pasa a qR. Si encuentra un 1 pasa a qPar.

Los movimientos de qPar, al igual que los de qImpar, son hacia la derecha, por lo que el tiempo de la máquina a lo sumo será 2N, por lo tanto O(N).

Entonces, si w pertenece a L, la cantidad de estados de la máquina <M> va a ser par, y la cantidad de 1 en la segunda cinta también, por lo tanto el recorrido va a terminar con qPar leyendo un blanco. En cambio, si w no pertenece a L, la cantidad de estados de la máquina <M> va a ser impar, y por lo tanto la cantidad de 1 en la segunda cinta va a ser también, por lo tanto el recorrido va a terminar con qImpar leyendo un blanco.

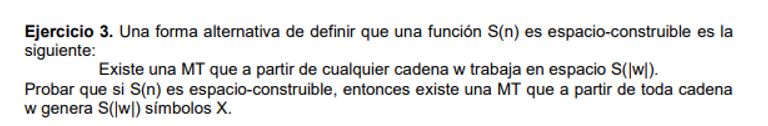


Como tenemos una MT que acepta un lenguaje, la idea es ir generando todos los strings de Σ\* y a cada uno evaluarlo con la máquina reconocedora, si esta para en qA, lo escribimos en la cinta output y avanzamos al siguiente string, en cambio, si para en qR avanzamos al siguiente string sin escribir nada.

La máquina reconocedora puede no parar, por eso debemos llevar un contador de los pasos para saber cuando entra en loop.



Como tenemos una MT que genera un lenguaje, la idea sería que por cada string generado, lo comparemos con el w, si son iguales la máquina reconocedora para en qA, sino generamos el próximo string del lenguaje y volvemos a realizar la comparación.



Asumimos que la MT original trabaja sobre una cinta para hacer más simple la definición.

La MT que genera símbolos X la vamos a llamar MX.

MX va a tener 2 cintas.

Por cada transición de la MT original definida como:

(qI, e) = (qJ, t, M)

Siendo qI y qJ cualquier estado y “M” cualquier movimiento de la MT original,, y siendo "e" y "t" cualquier símbolo de Γ

Se crean dos transiciones en MX tal que:

(qI, e, B) = (qJ, (t, M), (X, M))

(qI, e, X) = (qJ, (t, M), (X, M))

El cabezal de la segunda cinta se moverá a la par que el cabezal de la primera, con la diferencia de que la misma siempre escribirá un símbolo X, entonces cuando termine la ejecución de la MT habrá utilizado un espacio S(|w|), debido a que se recorrieron las mismas casillas que la MT original, y generará S(|w|) símbolos X, porque por cada movimiento de la cinta original escribimos una X en la segunda.



Se probará CLIQUE ≤c FCLIQUE.

Se cumple que existe una MT MFCLIQUE que trabaja en tiempo poly(n) y decide CLIQUE:

Dado una entrada w, MFCLIQUE hace los siguientes pasos:

1. Chequea que la entrada sea del formato (G, K) y que también sea válida.

2. Copia el grafo G en la cinta P.

3. Invoca al oráculo FCLIQUE.

4. Si el oráculo devuelve un grafo, se calcula la longitud, en caso contrario, rechaza.

5. Si la longitud es menor que K, se rechaza, en caso contrario, se acepta.

MFCLIQUE trabaja en tiempo poly(n), ya que el paso:

1. Lleva tiempo O(n) porque lee el input.

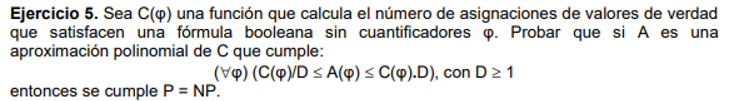
2. Lleva tiempo O(n) porque recorremos el input y lo pegamos en otra cinta.

3. Lleva tiempo constante.

4. Lleva tiempo O(n) porque consiste en leer el grafo resultante, que, como mucho, será igual de extenso como el original.

5. Lleva tiempo O(n) porque consiste en leer las longitudes y compararlas.

MFCLIQUE decide CLIQUE, porque si FCLIQUE encuentra un grafo de tamaño K o mayor, MFCLIQUE acepta. Si no hay un CLIQUE de tamaño K, es debido a que FCLIQUE nos devolverá un grafo de tamaño menor a K, o no devuelve ni siquiera un grafo, por lo tanto MFCLIQUE rechaza.



Para demostrar que P = NP, podemos decidir un lenguaje que esté en NPC.

Para este caso, decidiremos SAT.

Partimos con la hipótesis de que A es una aproximación polinomial de C:

La idea es que la máquina MTD MSAT va a decidir qué SAT haga uso de la función A.

Si una fórmula booleana no tiene asignación de valores de verdad, C(φ) va a dar 0, entonces A(φ) también va a dar 0.

Si A(φ) tiene cualquier otro valor es porque C(φ) es distinto de 0, entonces se puede asumir que existe al menos una asignación de valores de verdad que satisface la fórmula boolena.

MSAT hace lo siguiente:

1. Chequea que el input sea una fórmula booleana válida

2. Copia el input a una segunda cinta

3. Ejecuta A(φ)

4. Si el resultado de A(φ) es 0, rechaza, sino acepta.

MSAT trabaja en tiempo poly(n), ya que el paso:

1. Lleva tiempo O(n) porque consiste en recorrer el input.

2. Lleva tiempo O(n) porque para copiar recorremos el input y lo pegamos en otra cinta.

3. Lleva tiempo poly(n) porque A es una aproximación polinomial.

4. Lleva tiempo constante porque consiste en leer el string finito que devuelve A.

Se w pertenece a SAT, la MT MSAT reconoce SAT, esto quiere decir que existe alguna asignación de valores de verdad para la fórmula booleana que recibe, el valor de C(φ) será distinto de 0, por lo tanto el de A(φ) también, entonces la maquina aceptaría.

Si w no pertenece a SAT, la fórmula booleana no va a tener una asignación de verdad que la satisfaga, como C(φ) va a ser 0, A(φ) también lo será, entonces la máquina rechaza.



Si un lenguaje L está en P, entonces tiene una MTD MT que lo decide.

Para probar P ⊆ RP podemos crear una máquina RP con una rama de ejecución, en la cual se ejecute la MTD MT, entonces si esta acepta, todas las computaciones de RP lo hacen, por lo tanto la máquina RP acepta. En caso de que la MTD MT rechace, quiere decir que todas las computaciones de RP lo habrían hecho, entonces la máquina RP rechaza.

Como vemos RP se comportaría de la misma forma que la MTD MT y su tiempo de ejecución también es poly(n), por lo tanto queda probado que cualquier L en P estaría en RP.

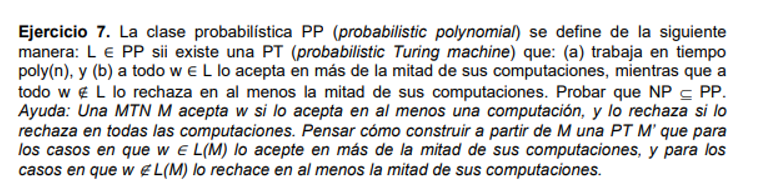
Si un lenguaje L pertenece a RP, entonces se decide con una máquina RP.

Para probar que RP ⊆ NP se puede simular una máquina RP con una MTN M que ejecuta una rama por cada una que ejecuta la máquina RP. Si RP acepta es porque, como mínimo, ½ de las computaciones aceptaron, entonces ½ de las ramas de la MTN aceptaran, por lo tanto M acepta. En caso de que RP rechace, todas las computaciones lo harán, entonces todas las ramas de la MTN también, por lo tanto M rechaza.

Debido a que la máquina RT ejecuta cada computación en tiempo poly(n), M también lo hará.

Como vimos, a cualquier lenguaje L que se encuentre en RP se le puede generar una MTN que lo decida, por lo tanto cualquier lenguaje que está en RP también lo está en NP.

Ahora que probamos que P ⊆ RP y RP ⊆ NP concluimos que P ⊆ RP ⊆ NP

.

Demostraremos que NP ⊆ PP

Si un lenguaje L pertenece a NP, entonces existe una MTN M que lo reconoce en tiempo poly(n). Construiremos una PT que reconozca en tiempo poly(n) a todo L de NP para así demostrar la inclusión.

Sabemos que las ramas de PT son iguales que las de MTN (con la diferencia de que paran en un estado u otro), entonces el tiempo de ejecución de cada una es poly(n), por lo tanto la máquina PT se ejecutará en tiempo poly(n).

Si tenemos una MTN definida, en la cual cuando su rama acepta, la del PT también, pero cuando rechaza, la rama de la PT aceptara o rechazara al azar, es decir, un 50% de probabilidades de aceptar, y un 50% de probabilidades de rechazar. Entonces por probabilidad, la mitad de las computaciones que en la MTN pararían en qR, en el caso de la PT lo harían en qA.

Si el L pertenece a NP, la MTN aceptará, teniendo como mínimo una de sus computaciones parando en qA, entonces el PT también, debido a que al menos la mitad más una computación pararían en qA.

En caso de que el L no pertenezca a NP la MTN rechazará, con todas sus computaciones parando en qR, entonces el PT también rechazará, debido a que la mitad de sus computaciones pararían en qA y la otra mitad en qR.

Si el L no pertenece a NP, la MTN lo va a rechazar, con todas sus computaciones parando en qR. La PT también va a rechazar, porque la mitad de sus computaciones van a parar en qA y la otra mitad en qR, y si esto ocurre PT rechaza.

Si el L pertenece a NP, la MTN lo va a aceptar, con al menos una de las computaciones parando en qA, por lo que la PT también lo va a aceptar, ya que la mitad más, al menos, una computación van a parar en qA (las ramas que paran en qR y se transforman a qA por el azar más las que originalmente paraban en qA).