Practica 3 creo

Sean Σ ={a,b} y L el conjunto de todos los lenguajes definidos sobre Σ. Diga si las

siguientes afirmaciones son verdaderas o falsas:

{λ} ∈ (L – CO-RE )

Acá todo depende si {λ} pertenece a CO-RE, ya que sabemos que {λ} pertenece a L .

Para que {λ} pertenezca a CO-RE entonces complemento de {λ} tiene que pertenecer a RE.

El complemento de {λ} son todos los elementos de Σ\* que no pertenecen a {λ}, llamémosle c{λ}.

Para que c{λ} pertenezca a RE entonces para c{λ} debe existir una MT que para todos los elementos de c{λ} los acepte.

¿Esta máquina existe?

La respuesta evidentemente es que si, puedes simplemente crear una máquina que espere cadenas de símbolos “a” y símbolos “b”, pero no acepta que la cadena no tenga símbolos, sus transiciones serían:

d(q0,B)=(qR,B,S)

d(q0,a)=(q1,a,D)

d(q0,b)=(q1,b,D)

d(q1,a)=(q1,a,D)

d(q1,b)=(q1,b,D)

d(q1,B)=(qA,B,D)

.: La MT existe.

.: c{λ} pertenece a RE.

.: {λ} pertenece a CO-RE.

.: La proposición es verdadera.

Esta bien esto?

RE ∪ R = L

Segun la teoria L – RE = ∅, además R no puede ser mas grande que RE, ni este que L. Por lo

tanto no L incluye de forma estricta a RE.

.: Es imposible que la proposición sea verdadera, ya que RE ∪ R está incluido estrictamente

en L.

Practica 4

7) Para los casos a), b) y c) del punto anterior ¿valen las recíprocas? Justifique.

b) L1 ∩ L2 ∈ RE ⇒ (L1 ∈ RE) AND (L2 ∈ RE)

Se me ocurre que es falso ya que es una intersección, y podría ser que L1 y L2 no son RE debido a los elementos que no tienen en común, mientras que L1 ∩ L2 si es RE, pero tampoco se logró encontrar cuales serian estos elementos como para escribir el contra ejemplo.

c) L1 ∪ L2 ∈ RE ⇒ (L1 ∈ RE) AND (L2 ∈ RE)

8) Si L es un subconjunto de un lenguaje recursivamente enumerable, ¿Puede afirmarse

entonces que L es recursivamente enumerable? Justifique.

Buscando en internet encontré que:

Nótese que los lenguajes recursivamente enumerables no son cerrados con la diferencia ni el complementario.

* L ? P puede no ser recursivamente enumerable

De esto entiendo que los subconjuntos de un lenguaje RE no necesariamente son RE, pero no hallo lenguaje RE al que quitando elementos deje de ser RE.

Esta bien?

Practica 5

5) Demuestre que LV ∉ RE

LV={(<M>)/L(M) = ∅}.

Considere que si <M> es un código inválido de máquina de Turing también pertenece a LV (ya que no reconoce ningún string). Así LV es el complemento del lenguaje LNV={(<M>)/L(M)≠∅}

(Ayuda: puede utilizar el complemento de Lu para encontrar una reducción)

LV={(<M>)/L(M) = ∅}

LNV=LV={(<M>)/L(M) ≠ ∅}

Lu = {(<M>,w) / M acepta w}

Lnu =Lu= {(<M>,w) / M no acepta w}

Podemos demostrar que existe una reducción Lnu α LV y como Lnu ∉ RE ⇒ LV ∉ RE

Vamos a demostrar que (Lnu α LV)

Para esto debe existir una funcion computable (o recursiva) f: Σ\*→ Σ\* tal que:

* f debe ser computable
* ∀(<M>,w) ∈ Σ\*, (<M>,w) ∈ Lnu⇔ <M’> ∈ LV.

f((<M>,w))=<M’>

Construiremos Mf para demostrar estos puntos

Mf trabaja de la siguiente manera:

Primero chequea que (<M>,w) sea un par válido, si no lo es entonces (<M>,w) ∉ Lnu por ende deberemos dejar una salida en la cinta tal que <M’> ∉ Lv, para esto borraremos la cinta y escribiremos el código correspondiente a una MT que acepta todo.

Si es un par valido, chequeamos que <M> sea un código válido de MT, si no lo es, lo dejaremos en la cinta y borramos w todo lo que no pertenezca a este, ya que por convención asumimos que corresponde a una MT M’ tal que <M> no acepta ningún input ((<M>,w) ∈ Lnu) y por ende L(M’) = ∅ (<M’> ∈ LV).

La parte importante viene si ambas cosas son válidas, en este caso Mf construye <M’> escribiendo las quintuplas necesarias para que M’ borre la entrada y escriba w en la cinta, posicione el cabezal y simula M sobre w. Así M’ para en qR o loopea para cualquier input ⇔ M no acepta w.

Puede demostrarse fácilmente que:

1. f es computable?

Creo que no pues Mf no se detiene luego de realizar una cantidad finita de acciones?

1. ¿∀(<M>,w) ∈ Σ\*, (<M>,w) ∈ Lnu ⇔ (<M’>,w) ∈ LV?
2. (<M>,w) ∈ Lnu ⇒ f((<M>,w)) ∈ LV.
   1. Si <M> no es un código válido de MT, por convención representa una MT que rechaza cualquier entrada (osea que (<M>,w) ∈ Lnu), en este caso se deja solo el código de M ya que sabemos que L(M)= ∅ (osea <M> ∈ LV).
   2. (<M>,w) ∈ Lnu ⇒ M rechaza w.

⇒ M para qR o loopea con input w.

⇒ M’ para en qR o loopea siempre con cualquier input

(por construcción)

⇒ (<M’>,w) ∈ LV.

b. (<M>,w) ∉ Lnu ⇒ f((<M>,w)) ∉ LV.

i. Si (<M>,w) no es un par valido entonces escribimos una MT

que acepta todo, por ende <M’> ∉ Lnu.

ii. (<M>,w) ∉ Lnu ⇒ M acepta w.

⇒ M para qA con input w.

⇒ M’ para en qA siempre con cualquier input

(por construcción)

⇒ (<M’>,w) ∉ LV.

Queria asegurarme que este punto este correcto

Practica 6

1 g) f : N → IR≥0, f(n) ∈ O(n) ⇒ 2f(n) ∈ O(2n).

Si f(n) ∈ O(n) eso implica que limn->inf f(n)/n=0 o limn->inf f(x)/n ∈ R+.

Sabemos que limn->inf 2f(n)/2n. Si ahora realizaremos el límite, nos quedaría 20/2inf o 2m/2inf, m ∈ R+ (un numero elevado al infinito da infinito si el numero es mayor a 1, y 0 se es menor a 1, si es 1 nos da indeterminado) en el primer caso 20/2inf=1/inf=0 y en el segundo 2m/2inf=w/inf=0, m,w ∈ R+.

La regla del límite nos dice entonces que 2f(n) ∈ O(2n) en ambos casos.

.: Tendríamos que N → IR≥0, f(n) ∈ O(n) ⇒ 2f(n) ∈ O(2n)

¿Está bien?¿Otra opción más simple no sería que c=2f(n) y listo?

i) Para todo polinomio p(n) de grado m, p(n) ∈ O(nm).

limn->inf p(n)/nm=limn->inf wnm+g(n)/nm, g(n) de grado menor a m, si sacamos factor común, lo máximo que podremos obtener será limn->inf nm-1(wn+...+q)/nm, simplificando nos quedamos con limn->inf(wn+...+q)/n, aplicando distributiva queda lim n->infwn/n+...+q/n, que es equivalente a limn->infw+...+q/n, remplazando es w+...+q/inf que es equivalente a w+...+0, lo que es w, w ∈ R+.

Con la regla del límite tendremos que p(n) ∈ O(nm).

.: Para todo polinomio p(n) de grado m, p(n) ∈ O(nm)

¿Está bien?

¿No hay forma más fácil de resolver este punto i y el f que sigue?

(Transitividad)

2-f) Si f(n) ∈ Ω(g(n)) y g(n) ∈ Ω(h(n)) ⇒ f(n) ∈ Ω(h(n))

Ω(f(n)) = {t:N → R≥0 / ∃ c ∈ R≥0, n0 ∈ N tq t(n) ≥ c f(n), n ≥ n0}

Deducimos lo siguiente:

* ∃ c ∈ R≥0, n0 ∈ N tq f(n) ≥ c g(n), n ≥ n0.
* ∃ c ∈ R≥0, n0 ∈ N tq g(n) ≥ c h(n), n ≥ n0.

Con esto podemos concluir que:

* ∃ c ∈ R≥0, n0 ∈ N tq f(n) ≥ c h(n), n ≥ n0.

¿Cómo? Pues siguiendo la siguiente explicación:

1. sabemos que f(n) ≥ c1 g(n).
2. ¿c1 g(n) ≥ c2 h(n)?

Evidentemente si, ya que sabemos que ∃ c ∈ R≥0, n0 ∈ N tq g(n) ≥ c h(n), n ≥ n0, si multiplicamos ambos lados por c1 tendremos que c1 g(n) ≥ c c1 h(n), lo cual podemos expresar como c1 g(n) ≥ c2 h(n), por ende seguimos cumpliendo con la definición de Ω.

1. Por Modus ponens entre 1 y 2 concluimos que f(n) ≥ c2 h(n), por ende llegamos a que

∃ c ∈ R≥0, n0 ∈ N tq f(n) ≥ c h(n), n ≥ n0. O sea que f(n) ∈ Ω(h(n)).

.: Concluimos que Si f(n) ∈ Ω(g(n)) y g(n) ∈ Ω(h(n)) ⇒ f(n) ∈ Ω(h(n))

¿Está bien?

Simetría:

g) f(n) ∈ Θ(g(n)) ⇐⇒ g(n) ∈ Θ(f(n)).

Θ(f(n)) = {t:N → R≥0 / ∃ c1, c2 ∈ R≥0, n0 ∈ N tq c1 f(n) ≤ t(n) ≤ c2 f(n), n ≥ n0}

Necesitamos demostrar 2 cosas para esto:

1. f(n) ∈ Θ(g(n)) ⇒ g(n) ∈ Θ(f(n)).
   1. si f(n) ∈ Θ(g(n))
   2. ⇒ ∃ c1, c2 ∈ R≥0, n0 ∈ N tq c1 g(n) ≤ f(n) ≤ c2 g(n), n ≥ n0.
   3. ⇒ ∃ c1, c2 ∈ R≥0, n0 ∈ N tq c1 f(n) ≤ g(n) ≤ c2 f(n), n ≥ n0.
   4. ⇒ g(n) ∈ Θ(f(n)).
2. g(n) ∈ Θ(f(n)) ⇒ f(n) ∈ Θ(g(n)) .
   1. si g(n) ∈ Θ(f(n))
   2. ⇒ ∃ c1, c2 ∈ R≥0, n0 ∈ N tq c1 f(n) ≤ g(n) ≤ c2 f(n), n ≥ n0.
   3. ⇒ ∃ c1, c2 ∈ R≥0, n0 ∈ N tq c1 g(n) ≤ f(n) ≤ c2 g(n), n ≥ n0.
   4. ⇒ f(n) ∈ Θ(g(n)).

.: f(n) ∈ Θ(g(n)) ⇐⇒ g(n) ∈ Θ(f(n)).

¿Está bien?

Practica 8

Ejercicio 6)

b)

p ← 0

for i ← 1 to n do

for j ← 1 to i do

for k ← 1 to n do

p ← p + 1

Primero intentare encontrar el tiempo de ejecución en función de n del algoritmo:

podemos decir que

t(n)= c1+sumatoria1aNconi(sumatoria1aIiconj(sumatoria1aINconk(c2))) =

= c1+sumatoria1aNconi(sumatoria1aIiconj(n\*c2)) =

= c1+(n\*c2)\*sumatoria1aNconi(sumatoria1aIiconj(1)) =

= c1+(n\*c2)\*sumatoria1aNconi(i\*1) =

= c1+(n\*c2)\*sumatoria1aNconi(i) =

= c1+(n\*c2)\*((n\*(n+1))/2) =

= c1+c2\*n\*(n2+n)/2) =

= c1+(c2/2)\*(n3+n2) =

= c1+(c2/2)\*n3+(c2/2)\*n2) =

= (c2/2)\*n3+(c2/2)\*n2) + c1

∴ Tenemos que el tiempo de ejecución de este algoritmo en funcion de n es t(n)= (c2/2)\*n3+(c2/2)\*n2) + c1.

Ahora debo buscar una f(n) tal que el tiempo de ejecución pertenezca a Θ(f(n)):

t(n) ∈ Θ(f(n)) sii t(n) ∈ O(f(n)) y t(n) ∈ Ω(f(n))

Se puede demostrar que t(n) ∈ Θ(f(n)) para f(n) siendo f(n)=n3 y t(n)=(c2/2)\*n3+(c2/2)\*n2) + c1..

Utilizando la regla de limites podemos hacer

((c2/2)\*n3+(c2/2)\*n2) / n3

->c2/2+(c2/2)/n

->c2/2+(c2/2)/inf

->c2/2+0 = c2/2 entonces ∈ R+

∴ t(n) ∈ Θ(f(n))

¿Está bien?

8) ¿Encuentra algún inconveniente para analizar las iteraciones while y repeat como recurrencias?

Lo único que logré encontrar en las teorías es que son complejas y existen algoritmos para los cuales no es posible hallar una funcion o definir una recurrencia.

9) Considerar las matrices A, B, C ∈ IR(n×n), y la notación tal que Xi,j , con 1 ≤ i, j ≤ 2 y X cualquiera de las matrices A, B o C, identifica una de las cuatro submatrices de orden n/2.

a) Dar el orden del tiempo de ejecución del algoritmo D&C que se describe con las ecuaciones

C1,1 = A1,1 × B1,1 + A1,2 × B2,1

C1,2 = A1,1 × B1,2 + A1,2 × B2,2

C2,1 = A2,1 × B1,1 + A2,2 × B2,1

C2,2 = A2,1 × B1,2 + A2,2 × B2,2

¿Sería necesario definir algo más?

Según la teoría hay que usar las recetas de recurrencia.

Con respecto a lo que falta definir es obvio que sí, falta definir el caso base, podría definirlo así:

| t(n)= c1 n≤1

<|

| t(n)= --- n>1

Para usar las recetas debemos identificar 3 cosas:

a cantidad de llamadas recursivas

b constante relacionada con la reducción de tamaño

k grado del polinomio de trabajo extra

En el algoritmo que nos dan podemos identificar:

a = 8

b = 2

k = 2 (por que el trabajo extra en este caso se basa en la suma de matrices cuadradas)

Aca serian 2 llamadas recursivas por cada ?

Así usando la receta para división de tamaño tenemos que:

a>bk osea 8>22 <-> 8>4 así que O(nlog b(a)) osea t(n) es orden O(nlog 2(8)) que haciendo las cuentas nos da que t(n) es O(n3)

∴ t(n) es O(n3)

¿Está bien?