Resumen CyC

Practica 1

Problema Computacional

Consiste en una especificacion de un conjunto de datos de entrada y datos de salida correspondientes a los ingresados.

Tenemos

- ➔ Conjunto de Problemas Computacionales PC
- ➔ Conjunto de Problemas Computacionales Computables PCC

Un problema puede tener una o varias instancias (valores particulares que toma la entrada)

PCC (Problemas Computacionales Computables) = R

PC = RE U CO-RE

PCC ⊆ PC

Lógica proposicional

Es una oración/afirmación con valor declarativo o informativo, de la cual se puede predicar su verdad o falsedad.

Operaciones:

Operaciones: operadores y operandos

– Negación ==> Unaria

– Conjunción

– Disyunción ==> Binarias

– Condicional

Tabla de verdad del Condicional

p	q	p→q
V	V	V
V	F	F
F	V	V
F	F	V

Equivalencia lógica.

Se dice que dos proposiciones son lógicamente equivalentes, o simplemente equivalentes. Si coinciden sus resultados para los mismos valores de verdad. Se indican como p ⇔ q.

Propiedades del Condicional

Un condicional y su recíproca no son lógicamente equivalentes (distintas tablas de verdad). Se dice que $q \rightarrow p$ es el recíproco de $p \rightarrow q$.

Un condicional y su contrarrecíproca son lógicamente equivalentes (iguales tablas de verdad). El contrarrecíproco del enunciado $p \rightarrow q$ es $\sim q \rightarrow \sim p$

$$p \rightarrow q \Leftrightarrow \sim q \rightarrow \sim p$$

~p v q <=> p -> q

p -> q <=> ~q -> ~p

Puede que q->p y p->q sean verdaderos en ese caso entonces p <-> q. Este queda definido como (p->q) ^ (q->p)

Tabla de verdad

p	**q**	**p↔q**
V	V	V
V	F	F
F	V	F
F	F	V

Tautologia → proposición (compuesta) que es cierta para todos los valores de verdad de sus variables.

Observar que si $p \leftrightarrow q$ es una tautología entonces p y q son lógicamente equivalentes, lo que se puede expresar como $p \Leftrightarrow q$

De la misma manera, cuando $p \rightarrow q$ es una tautología decimos que existe una implicación lógica que la expresamos $p \Rightarrow q$

Tautologias mas importantes

1.- Doble negación.

a). $\sim\sim p \Leftrightarrow p$

2.- Leyes conmutativas.

a). $(p \vee q) \Leftrightarrow (q \vee p)$

b). $(p \wedge q) \Leftrightarrow (q \wedge p)$

c). $(p \leftrightarrow q) \Leftrightarrow (q \leftrightarrow p)$

3.- Leyes asociativas.

a). $[(p \vee q) \vee r] \Leftrightarrow [p \vee (q \vee r)]$

b). $[(p \wedge q) \wedge r] \Leftrightarrow [p \wedge (q \wedge r)]$

4.- Leyes distributivas.

a). $[p \vee (q \wedge r)] \Leftrightarrow [(p \vee q) \wedge (p \vee r)]$

b. $[p \wedge (q \vee r)] \Leftrightarrow [(p \wedge q) \vee (p \wedge r)]$

5.- Leyes de idempotencia.

a). $(p \vee p) \Leftrightarrow p$

b). $(p \wedge p) \Leftrightarrow p$

6.- Leyes de Morgan

a). $\sim(p \vee q) \Leftrightarrow (\sim p \wedge \sim q)$

b). $\sim(p \wedge q) \Leftrightarrow (\sim p \vee \sim q)$

7.- Contrapositiva.

a). $(p \rightarrow q) \Leftrightarrow (\sim q \rightarrow \sim p)$

8.- Implicación.

a). $(p \rightarrow q) \Leftrightarrow (\sim p \vee q)$

b). $(p \rightarrow q) \Leftrightarrow \sim(p \wedge \sim q)$

9.- Equivalencia

a). $(p \leftrightarrow q) \Leftrightarrow [(p \rightarrow q) \wedge (q \rightarrow p)]$

10.- Adición.

a). $p \Rightarrow (p \vee q)$

11.- Simplificación.

a). $(p \wedge q) \Rightarrow p$

12.- Modus ponens.

a). $[p \wedge (p \rightarrow q)] \Rightarrow q$

13- Modus tollens.

a). $[(p \rightarrow q) \wedge \sim q] \Rightarrow \sim p$

14.- Transitividad del $\leftrightarrow$ y del $\rightarrow$

a). ¿Cómo sería?

Las tautologías relacionadas con $\Leftrightarrow$ y $\Rightarrow$ permiten “derivar” o “generar” o “tratar” información nueva o distinta

Esquemas proposicionales

Se llama esquema proposicional en la indeterminada x a toda expresión que contiene a x y tiene la propiedad que existe por lo menos un nombre tal que la expresión obtenida sustituyendo la indeterminada por dicho nombre, es una proposición. Estas indeterminadas se llaman variables o incógnitas y se utilizan símbolos tales como P(x), Q(x), F(x), para designar esquemas de incógnita x.

UNIVERSAL: $(\forall n):(P(n))$

EXISTENCIAL: $(\exists n):(P(n))$

La proposición con el operador universal (∀n):(P(n)) será V si todos los valores posibles de n hacen V la proposición resultante del esquema. Será F en cc.

La proposición con el operador universal (∃n):(P(n)) será V si hay al menos un valor posible de n que hace V la proposición resultante del esquema. Será F en cc.

Expresiones sinónimas:

$$\sim(\forall n)\text{: P}(n) \text{ y } (\exists n)\text{: }\sim\text{P}(n)$$

$$\sim(\exists n)\text{: P}(n) \text{ y } (\forall n)\text{: }\sim\text{P}(n)$$

Conjuntos y Cardinalidad

Colección o agrupación de objetos que es tratada como una unidad. Estos objetos se llaman elementos, no tienen orden y no se repiten.

Colección sin elementos es → conjunto vacío ∅

∅: Sin elementos

{∅} es distinto a vacio, puesto que tiene un elemento.

Ejemplo:

- ∅ = { }: Conjunto vacío, sin elementos

- A = {∅}, tiene 1 elemento, que es el ∅

- A = {1, {3,-1}, ∅}: 3 elementos: 1, {3,-1}, ∅

Ni 3 ni -1 son elementos de A, son elementos del conjunto {3,-1}, que es un elemento del conjunto A.

X ∈ ∅ siempre es falsa

Formas de expresión:

- Diagrama de Venn (gráfico)
- Enumeración A = {1, 2, 3, 4, 5}
- Para los infinitos:
- N = {x / x es un número natural} es el conjunto de los números naturales
- - P = {n / n es par} = {n / n = 2k con k ∈ N}

Para todo x, x no pertenece a vacio.

Subconjuntos:

A es subconjunto o parte de B si se cumple que

$$(\forall x):(x \in A \rightarrow x \in B)$$

Se dice que A esta incluido en B o que es igual $A \subseteq B$

Si B tiene al menos un elemento que no esta en A se dice que

$$(\forall x):(x \in A \rightarrow x \in B) \wedge (\exists x):(x \in B \wedge x \notin A)$$

$A \subset B$

$\varnothing \subseteq \{a, b\}$, Def. de inclusión: $A \subseteq B \Leftrightarrow (\forall x):(x \in A \rightarrow x \in B)$

$A = \varnothing$, $B = \{a, b\}$

Debería demostrarse que $(\forall x):(x \in A \rightarrow x \in B)$, y más específicamente $(x \in A \rightarrow x \in B)$

$x \in A$ en este caso es $x \in \varnothing$, y por la definición de $\varnothing$, $(\forall x):(x \notin \varnothing)$, es decir que $x \in \varnothing$ es F para todo x.

Por lo anterior, y por el operador condicional, se tiene que $x \in A \rightarrow x \in B$ es V para cualquier (para todo) valor posible de x, independientemente del conjunto B que sea, y en particular para $B = \{a, b\}$

Por lo que queda demostrado que $\varnothing \subseteq \{a, b\}$

Igualdad entre conjuntos

$$(\forall x):(x \in A \leftrightarrow x \in B)$$

Observación: $(A = B) \Leftrightarrow ((A \subseteq B) \wedge (B \subseteq A))$

Operaciones básicas de Conjuntos

Intersección: $A \cap B = \{x / x \in A \wedge x \in B\}$

Unión: $A \cup B = \{x / x \in A \vee x \in B\}$

Diferencia: $A - B = \{x / x \in A \wedge x \notin B\}$

Complemento: Si $A \subseteq B$, $\overline{A}_B = B - A$

Si B es el conjunto universal se denota $\overline{A}$ y se puede escribir como $\{ x / x \notin A \}$

Equivalencias de conjuntos

Leyes conmutativas:

$$A \cup B = B \cup A \text{ y } A \cap B = B \cap A$$

Leyes distributivas:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

Leyes de Morgan:

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

¿Por qué se llamarán leyes de Morgan?

Ley de doble complemento:

$$\overline{\overline{A}} = A$$

Probar que $(A - B) \cap (A - C) = A - (B \cup C)$

Dem.

	$x \in (A - B) \cap (A - C)$	
$\Leftrightarrow$	$x \in (A - B) \wedge x \in (A - C)$	(def. de intersec.)
$\Leftrightarrow$	$(x \in A \wedge x \notin B) \wedge (x \in A \wedge x \notin C)$	(def. de resta)
$\Leftrightarrow$	$x \in A \wedge \sim(x \in B) \wedge \sim(x \in C)$	(Lóg. Prop.)
$\Leftrightarrow$	$x \in A \wedge \sim(x \in B \vee x \in C)$	(Morgan)
$\Leftrightarrow$	$x \in A \wedge \sim(x \in (B \cup C))$	(def. de unión)
$\Leftrightarrow$	$x \in A \wedge x \notin (B \cup C)$	(def. de $\notin$)
$\Leftrightarrow$	$x \in A - (B \cup C)$	(def. de resta)

Asoc.
Conm.
simp.

Por lo tanto $(A - B) \cap (A - C) = A - (B \cup C)$

1) Probar la siguiente ley distributiva A u (B n C) = (A u B) n (A u C)

Sea X ∈ A u (B n C)

$\Leftrightarrow$ $x \in A \vee (x \in B \wedge x \in C)$ def. unión e intersección.

$\Leftrightarrow$ $(x \in A \vee x \in B) \wedge (x \in A \vee x \in C)$ distribuyo la disyunción $x \in A$ sobre la conjunción $x \in B \wedge x \in C$. Lógica prop.

$\Leftrightarrow$ $x \in (A \text{ u } B) \text{ n } (A \text{ u } C)$ def. unión e interseccion

https://www.youtube.com/watch?v=E0oZCUld4_s

2) Probar la siguiente ley de Morgan: El Complemento de A unión B es igual al complemento de A intersección el complemento de B

Sea $X \in (A \text{ u } B)^C$
$\Leftrightarrow$ $X \in/ (A \vee B)$ def. unión.
$\Leftrightarrow$ $(X \in/ A) \wedge (x \in/ B)$ Lógica prop.
$\Leftrightarrow$ $X \in A^c \wedge X \in B^c$
$\Leftrightarrow$ $X \in (A^c \text{ n } B^c)$ def. intersección.

https://www.youtube.com/watch?v=rvoHLxCPohc

https://www.youtube.com/watch?v=Qne9XD0FAl8

3) Probar que el complemento del complemento de A es igual a A

Sea $X \in (A^C)^C$
$\Leftrightarrow$ $X \in/ A^C$
$\Leftrightarrow$ $X \in A$

Producto cartesiano

{(x,y)/ x en A, y en B}

A={1}

A^5={(1,1,1,1,1)}=AxAxAxAxA

{}^5={}

{}xA={}

B={2,3}

C={4,5}

BxC={(2,4),(2,5),(3,4),(3,5)}

|B|+|C|=|BxC|

Función: establecen una relación entre dos conjuntos, donde f: A → B define que un elemento de A (dominio) está relacionado a lo sumo con 1 y solo 1 elemento de B (codominio). Se puede especificar como un subconjunto de A x B

$f : N \to N$, $f(n) = 2n$
$\{(0, 0), (1, 2), (2, 4), (3, 6), \ldots\} = \{(n, 2n), n \in N\}$

$f: R \to R$, $f(x) = -(x^{1/2})$ $\quad$ $x^{1/2}$ no es una función
$\{(0, 0), (4, -2), (9, -3), (16, -4), \ldots\}$, no está definida para $x < 0$
(ejemplos solamente de N, pero hay pares de reales (x, y) con $x \geq 0$)

$f: R^+ \to R$, $f(x) = +(x^{1/2})$ $\quad$ dominio de reales > 0
$\{ (4, 2), (9, 3), (16, 4), \ldots\}$ $\quad$ (0, 0) no pertenece a este conjunto, ni ningún par (x, y) con $x \leq 0$

Conjunto de partes

El conjunto de partes de A es el conjunto formado por todos los subconjuntos de A

$\rho(A) = \{B / B \subseteq A\}$

$A = \{a,b\}$

$\rho(A) = \{\{\}\{a\}\{b\}\{a,b\}\}$

¿$\rho(\varnothing)$? $\quad$ $\rho(\varnothing) = \{\varnothing\} = \{B / B \subseteq \varnothing\}$

¿$\rho(\{\varnothing\})$?

¿$\rho(\{\{\varnothing\}, \varnothing\})$?

En todos los casos, conviene identificar los elementos de los conjuntos, en el caso anterior, se puede hacer como "intermedio"

$A = \{\{\varnothing\}, \varnothing\}$, denotando $a = \{\varnothing\}$ y $b = \varnothing$, y quedaría

$\rho(\{a, b\}) = \{\varnothing, \{a\}, \{b\}, \{a,b\}\}$ y luego reemplazar a y b

$\rho(\{a, b\}) = \{\varnothing, \{\{\varnothing\}\}, \{\varnothing\}, \{\{\varnothing\}, \varnothing\}\}$

NOTITA HERMOSA: El elemento vacio esta en todos los conjuntos.

{1, 2} x {a} x {a, b}

A= {1,2}

B={a}

C = {a,b}

A X B X C= {(w,z,j) / w ∈ A ∧ z ∈ B ∧ j ∈ C} = {(1,a,a), (1,a,b), (2,a,a), (2,a,b)}

Cardinalidad

Si A es un conjunto finito, la cardinalidad es el numero de elementos de A.

$$|A| = n \Rightarrow |\rho(A)| = 2^n$$

Caso Base)

n = 0
A = ∅
$\rho(A) = \rho(\emptyset) = \{\emptyset\} \Rightarrow |\rho(A)| = |\rho(\emptyset)| = |\{\emptyset\}| = 1 = 2^0 = 2^n$

Hi.)

$|A| = n \Rightarrow |\rho(A)| = 2^n$

Dem.)

$|A'| = n+1 \Rightarrow |\rho(A')| = 2^{n+1}$
$A' = \{a_1, \ldots, a_n, a_{n+1}\} = \{a1, \ldots, a_n\} \cup \{a_{n+1}\} = A \cup \{a_{n+1}\}$
$\rho\{A\} = \{ \emptyset, \{a_1\}, \ldots, \{a_n\}, \{a_1, a_2\}, \ldots, \{a_1, \ldots, a_n\}\}$, $|\rho\{A\}| = 2^n$ por Hi

Todos los subconjuntos de A son subconjuntos A', porque todos los elementos de A son elementos de A' (subconjunto de A = elem. de ρ{A})
Para determinar la cantidad de elementos de ρ{A'} se deben agregar ("faltan") los subconjuntos de A' que contienen al elemento a_{n+1}:
$\{ \emptyset, \{a_1\}, \ldots, \{a_n\}, \{a_1, a_2\}, \ldots, \{a_1, \ldots, a_n\}\} \cup \{a_{n+1}\} \cup \{a_{n+1}\} \cup \{a_{n+1}\} \cup \{a_{n+1}\} \cup \{a_{n+1}\}$

Se puede pensar como que los subconjuntos que "faltan" se pueden formar con los subconjuntos de $A \cup \{a_{n+1}\}$
Por lo tanto, la cantidad total sería el doble de elementos de ρ{A}, y como $|\rho\{A\}| = 2^n$ se tiene que $|\rho\{A'\}| = 2x2^n = 2^{n+1}$, que es lo que se debía demostrar.

p = rho

A={1,2}

p(A)={{},{1},{2},{1,2}}

|A|<|p(A)|

|p(A)|=2^|A|

si A es infinito contable, p(A) es infinito incontable

¿Cuál es el cardinal de A X B si |A| = n y |B| = m? → |A X B| = n x m

Cardinalidad en conjuntos infinitos

Definicion. Se establece que la cardinalidad de un conjunto A es menor o igual que la cardinalidad de un conjunto B, y se denota $|A| \leq |B|$ sí y sólo si se puede establecer una correspondencia "uno a uno" de cada elemento de A con un elemento distinto de B

$$|A| \leq |B| \Leftrightarrow (\exists f): (f \text{ es una función inyectiva de A en B})$$

Puede demostrarse que es una relación de orden (reflexiva, antisimétrica y transitiva)

Definición. $|A| = |B| \Leftrightarrow |A| \leq |B| \wedge |B| \leq |A|$

Definición. $|A| < |B| \Leftrightarrow |A| \leq |B| \wedge |B| \neq |A|$

a) Mostrar que $|N| = |N^+|$

1) $|N| \leq |N^+|$ $\qquad$ f_1**:** $N \rightarrow N^+$ **iny.**
2) $|N^+| \leq |N|$ $\qquad$ f_2**:** $N^+ \rightarrow N$ **iny.**

Cuando un conjunto es subconjunto del otro la función inyectiva es "trivial": la identidad. En este caso:
$|N^+| \leq |N|$ **seguro es V por Id:** $N^+ \rightarrow N$**, Id(n) = n es iny.**

Faltaría demostrar 1), que no es demasiado difícil, ej:
Id: $N \rightarrow N^+$**, Succ(n) = n+1 es iny.**

Con lo cual están demostrados 1) y 2) y por lo tanto

$$|N| = |N^+|$$

Si bien es "intuitivo" pensar que N "tiene un elemento más" que N +, en realidad la idea de "uno más" no puede relacionarse con los infinitos elementos de ambos conjuntos. De hecho, según la noción de cardinalidad que consideramos, |N| = |N +|, tal como fue demostrado

b) Mostrar que $|P| = |N|$ con $P=\{n/n$ es un número par$\}$

1) $|P| \leq |N|$ Id: $P \rightarrow N$, $Id(n) = n$ es inyectiva
2) $|N| \leq |P|$ dbl: $N \rightarrow P$, $dbl(n) = 2n$ es inyectiva

En este caso, aunque "intuitivamente" P tiene "la mitad" de elementos que N en realidad no existe la noción de "mitad de infinito". Intuitivamente tenemos un problema más relacionado con nuestra forma de entender los conjuntos infinitos que con la noción o definición formal de cardinalidad de conjuntos infinitos

$$|N| = |NxN| \Leftrightarrow (\ |N| \leq |NxN| \ \wedge \ |NxN| \leq |N|\ )$$

a) $|N| \leq |NxN|$ $\quad$ $Idd: N \rightarrow NxN$, $Idd(n) = (n,n)$

b) $|NxN| \leq |N|$, $\quad$ $f: NxN \rightarrow N$, f inyectiva

Con $f(i,j) = \frac{(i+j)\,(i+j+1)}{2} + i$ se tiene esa función

$$\underbrace{(0,0)}_{1} \underbrace{(0,1)\ (1,0)}_{2} \underbrace{(0,2)\ (1,1)\ (2,0)}_{3} \underbrace{(0,3)\ (1,2)\ (2,1)\ (3,0)}_{4}$$

0 1 2 3 4 5

Y a partir de esa cantidad, la posición estará dada por "i"

Si |A| <= |N| entonces A es infinito CONTABLE.

p(N) es no contable.

Se puede representar una computadora como una funcion N->N

El conjunto de todos los programas de computacion que puede escribirse para computar funciones es contables pq cada programa puede verse como una secuencia de 1 y 0.

- Mostrar que un conjunto puede tener la misma cardinalidad que un subconjunto propio de sí mismo.

P = {p/p ∈ N, p es un numero par}

|P| = |N|

19.1) |P| <= |N|

19.2) |N| <= |P|

Para probar 19.1) es sencillo mediante la función de identidad. Para probar 19.2) podemos utilizar la función inyectiva dbl: N → P, dbl(n) = 2n

Está demostrado que

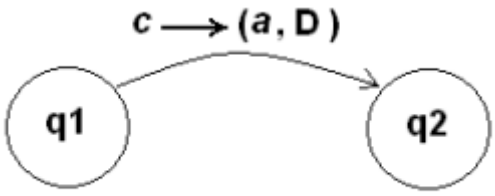
- Si $\exists$ f_1: A $\rightarrow$ B y f_2: B $\rightarrow$ A inyectivas $\Rightarrow$ $\exists$ f_3: A $\rightarrow$ B biyectiva (Cantor-Schröder-Bernstein)

Practica 2

Máquina de Turing

En cada instante, la máquina se encuentra en algún estado qi , perteneciente al conjunto finito Q de todos los estados posibles Q={q0 ,q1 ,q2 , … qn }

- Siempre comienza en q0
- Si existe una cadena de símbolos de entrada, la máquinacomienza apuntando al primer símbolo de esta cadena.
- Si no existe una cadena de entrada escrita en la cinta, sólo hay símbolos “B” en cada celda de la misma.
- La cadena de entrada estará limitada por infinitos B a izquierda y derecha. Además no hay ningún símbolo B en medio de la cadena
- Comportamiento definicido por función de transición



Si en el estado q1 hay una c entonces deja una a y se mueve a la derecha.

Máquina de Turing de/para Cómputo

Definición. Una Máquina de Turing de Cómputo se puede definir con una quíntupla

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0 \rangle$$

tal que:

Q es un conjunto finito de estados de M

Σ es el alfabeto de la entrada

Γ es el alfabeto de la cinta. $\Sigma \subset \Gamma$ y $B \in (\Gamma - \Sigma)$

q_0 es el estado inicial de M ($q_0 \in Q$)

δ es la función de transición de M.

Se define δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{D, I\}$,

D e I representan el movimiento del cabezal a derecha e izquierda respectivamente.

EL "resultado" del cómputo es lo que queda en la cinta

Programa MT NO es un programa secuencial sino que una funcion matematica de transicion

Si no existe transicion para el estado y simbolo se detiene

Alternativo con estado Qd

Modelo alternativo: MT con estado de detención q_d

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_d \rangle$$

Q es un conjunto finito de estados de M

Σ es el alfabeto de la entrada

Γ es el alfabeto de la cinta. $\Sigma \subset \Gamma$ y $B \in (\Gamma - \Sigma)$

q_0 es el estado inicial de M ($q_0 \in Q$)

q_d es el estado de detención de M ($q_d \notin Q$)

δ es la función de transición de M.

Se define δ: $Q \times \Gamma \rightarrow Q \cup \{q_d\} \times \Gamma \times \{D, I\}$

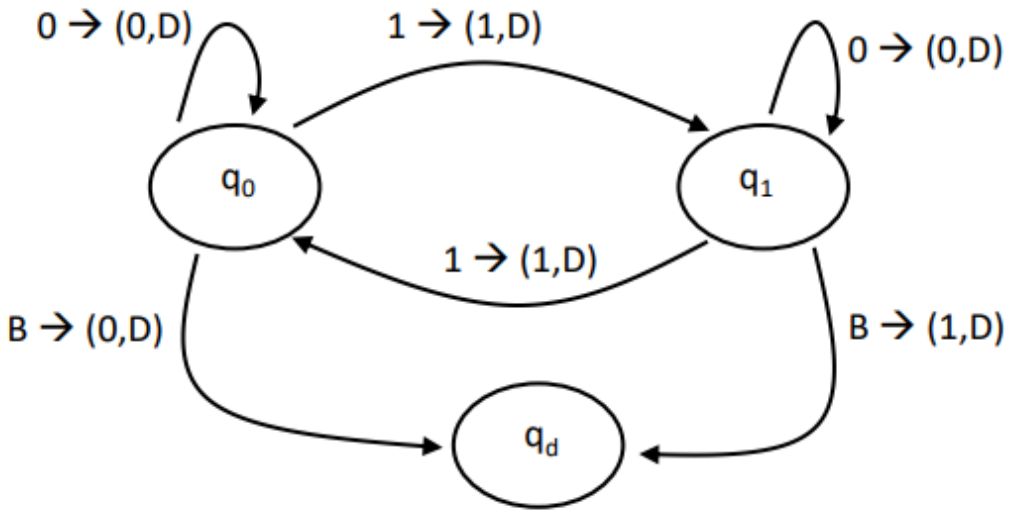
Necesariamente la máquina dejará de computar cuando llegue a q_d porque no puede aplicar δ (q_d no es *parte del dominio*)

Ahora tienen que estar todas las transiciones a qD si quiere terminar

Ejemplo distintas notaciones

$$M = <\{q_0,q_1\}, \{0,1\}, \{0,1,B\}, \delta, q_0, q_d>$$

Grafo de δ



$$M = <\{q_0,q_1\}, \{0,1\}, \{0,1,B\}, \delta, q_0, q_d>$$

Notación funcional de δ

$$\delta: Q \times \Gamma \rightarrow Q \cup \{q_d\} \times \Gamma \times \{D, I\}$$

$\delta(q_0,0) = (q_0,0,D)$

$\delta(q_0,1) = (q_1,1,D)$

$\delta(q_0,B) = (q_d,0,D)$

$\delta(q_1,0) = (q_1,0,D)$

$\delta(q_1,1) = (q_0,1,D)$

$\delta(q_1,B) = (q_d,1,D)$

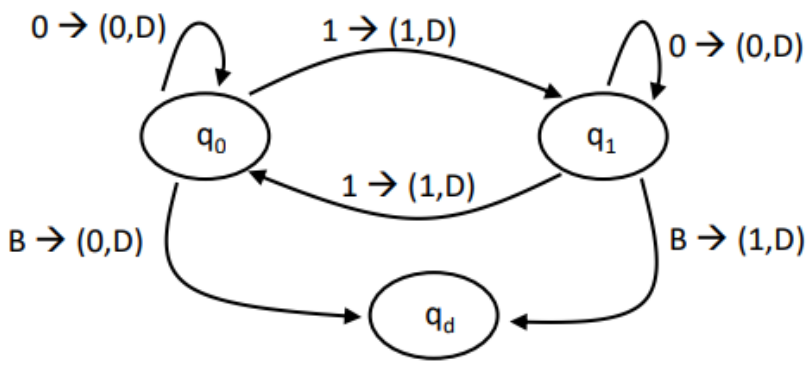
Descripción Instantánea de una MT y Traza

..s1s2 q s3s4..

Significa que está en el estado q leyendo s3

Traza: Secuencia de movimientos

Con |- indica que la MT M pasa en un solo movimiento de una configuración (instantánea) a otra. Con |-* es en cero o mas pasos.



Para la MT M del grafo, con la entrada 100101

$q_0100101 \vdash_M 1q_100101 \vdash_M 10q_10101 \vdash_M 100q_1101 \vdash_M$

$1001q_001 \vdash_M 10010q_01 \vdash_M 100101q_1B \vdash_M 1001011q_dB$

Lenguaje: conjuntos de sentencias (strings o cadenas) construidas a partir de un conjunto finito de símbolos (el alfabeto). Es una secuencia FINITA

Definición. Alfabeto: Diremos que un conjunto finito S es un alfabeto si $\Sigma \neq \varnothing$ y $(\forall x)(x \in \Sigma \rightarrow$ x es un símbolo indivisible)

Definicion. Palabra: Se dice que w es una palabra (cadena, sentencia o string) sobre Σ si w es una secuencia finita de símbolos de Σ

Longitud de palabra → |w| → número de símbolos que contiene w

λ es la cadena nula → |λ| = 0

Σ* → conjunto de todas las palabras formadas por símbolos de Σ (incluido λ)

Ejemplo: Σ = {a,b} Σ * = { λ, a, b, aa, ab, ba, bb, aaa…}

Concatenacion → v.w → o vw → Es asociativa pero no conmutativa

|v.w| = |v| + |w|

λ.v = v. λ = v

Potencia i-esima de w → w0 = λ

$w(i+1) = w.w^i$

si w = ab, w3 = ababab

Un lenguaje definido sobre Σ es

cualquier subconjunto de Σ^*

Ejemplo: si $\Sigma = \{0,1\}$
$\varnothing$, Σ^* ,$\{\lambda\}$, $\{w \in \Sigma^* \,/\, w$ comienza con 1$\}$ son lenguajes sobre Σ

Si L es un lenguaje sobre Σ, su complemento también lo es (Complemento de L respecto de Σ^* es $\bar{L} = \Sigma* - L$)

Nota: Llamaremos $\mathscr{L}$ al conjunto de todos los lenguajes definidos sobre el alfabeto Σ, es decir:

$$\mathscr{L} = \rho(\Sigma^*)$$

Máquinas de Turing Reconocedoras

Hay dos estados de paradas:

- qA : el estado de aceptación.
- qR : el estado de rechazo.

Se define d de manera completa, considerando todos los estados de Q. Sin embargo, ni qA ni qR pertenecen a Q.

Se puede rechazar con qR o loopeando. Lo importante no es lo que queda en la cinta, si no el estrado en el que se termina.

$M = <Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R>$ con $q_A, q_R \notin Q$

Donde:

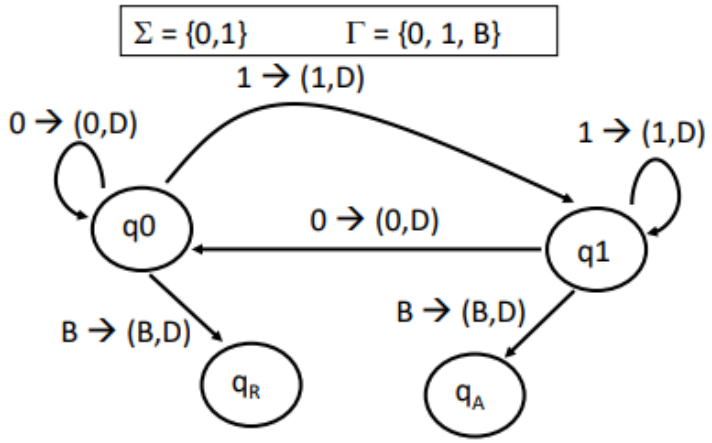
Q, Σ, Γ y q_0 se definen como en el caso de las MT de cómputo

$\delta: Q \times \Gamma \rightarrow Q \cup \{q_A, q_R\} \times \Gamma \times \{D, I\}$, completa

q_A es el estado de aceptación, q_R es el estado de rechazo

M se detiene $\Leftrightarrow$ M pasa al estado q_A o q_R

Ejemplo (revisitado)



$M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$

$Q=\{q0,q1\}$ $\Sigma=\{0,1\}$ $\Gamma=\{B,0,1\}$

$\delta: Q \times \Gamma \rightarrow Q \cup \{q_A, q_R\} \times \Gamma \times \{D, I\}$

$\delta(q0,0)=(q0,0,D)$ $\delta(q0,1)=(q1,1,D)$ $\delta(q0,B)=(q_R,B,D)$

$\delta(q1,0)=(q0,0,D)$ $\delta(q1,1)=(q1,1,D)$ $\delta(q1,B)=(q_A,B,D)$

Otra forma de especificar la función de transición δ es con una tabla

δ	0	1	B
q0	(q0,0,D)	(q1,1,D)	(q_R,B,D)
q1	(q0,0,D)	(q1,1,D)	(q_A,B,D)

Ejemplo - Traza

δ	0	1	B
q0	(q0,0,D)	(q1,1,D)	(q_R,B,D)
q1	(q0,0,D)	(q1,1,D)	(q_A,B,D)

Traza de ejecución (todos los movimientos) de la MT que reconoce secuencias binarias terminadas en 1 para la entrada *w*=0101

$$q_0 0101 \vdash_M 0q_0 101 \vdash_M 01q_1 01 \vdash_M 010q_0 1 \vdash_M 0101q_1 B \vdash_M 0101Bq_A B$$

Se puede escribir entonces:

$$q_0 0101 \vdash^*_M 0101Bq_A B$$

Lenguajes aceptados por la MT

L(M)= palabras del conjunto de todas las palabras que terminan en qA

w esta en el lenguaje de la maquina si con esa entrada M para en Qa

Algunas bellas anotaciones de la practica:

L1 = Ø L2 = {λ}

L1 es un conjunto vacio (sin elementos) y L2 es el conjunto cuyo elemento es una cadena vacia (que es un elemento valido)

¿Puede el alfabeto de la cinta (Γ) ser el mismo que el alfabeto de entrada (Σ)?

No, ya que el alfabeto de la cinta siempre tiene un elemento de mas que es B y Σ no puede tenerlo

Mostrar que Σ* es infinito contable.

|Σ*| <= |N| se puede probar con la función inyectiva f: Σ* → N en donde se le asigna a cada cadena de Σ* a un número natural ordenándolas primero por su longitud y luego enumerando las cadenas de la misma longitud en orden lexicográfico (orden alfabético).

¿Cuáles de los siguientes conjuntos son lenguajes definidos sobre Σ? Ø, Σ, Σ*, {λ}, {λ} ∪ Σ, {Ø}

Todos menos {Ø} ya que este no es un subconjunto de Σ* (Ø si es subconjunto pero {Ø} no)

Sintaxis y Semantica

Sintaxis responde si los simbolos están combinados de forma correcta semántica responde al significado de la sentencia o sentido

Practica 3

Modelos Equivalentes

Existen numerosos modelos de MT que se han probado equivalentes

- Dos Máquinas de Turing son equivalentes si y solo si L(M1)=L(M2)
- Dos modelos de MT son equivalentes si para cada MT de un modelo existe una MT equivalente en el otro modelo.

Para demostrar que dos modelos son equivalentes hay que básicamente:

Para toda MT M del modelo X existe una MT M´ del modelo Y equivalente y viceversa

Para demostrar que dos máquinas de Turing son equivalentes (L(M1)=L(M2)) hay que demostrar:

1. $L(M) \subseteq L(M')$
2. $L(M') \subseteq L(M)$ → Usaremos contrarecíproca ($w \notin L(M) \Rightarrow w \notin L(M')$)

Algunos modelos de MT equivalentes con los ya mencionados.

Modelo D-I-S (Derecha-Izquierda-Sin movimiento)

Admite transiciones sin movimiento del cabezal de la cinta.

$M = <Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R>$ con $Q, \Sigma, \Gamma, q_0, q_A, q_R$, definidos como en el modelo de referencia (que llamaremos modelo D-I en este caso)

y $\delta: Q \times \Gamma \rightarrow Q \cup \{q_A, q_R\} \times \Gamma \times \{D, I, S\}$

(S significa sin movimiento)

Modelo de MT de k cintas

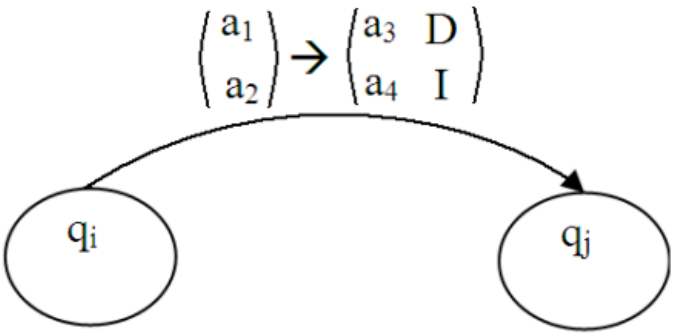
- Consiste en un control con k cintas y k cabezales que pueden moverse en forma independiente.
- La entrada se encuentra en la primera cinta y todas las demás están en blanco.

$M = <Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R>$

con $Q, \Sigma, \Gamma, q_0, q_A, q_R$ definidos como en el modelo estándar D-I-S de una cinta

y $\delta: Q \times \Gamma^k \rightarrow Q \cup \{q_A, q_R\} \times (\Gamma \times \{D, I, S\})^k$

Ejemplo: $\delta(q_i,(a_1,a_2)) = (q_j,(a_3,D),(a_4,I))$



Estando en el estado q_i, al leer a_1 en la primera cinta y a_2 en la segunda, escribe a_3 en la primera y a_4 en la segunda, mueve a la derecha el cabezal de la primera cinta y a la izquierda en de la segunda cinta.

Ejercicio: Definir la δ de transición de una MT con 2 cintas que reconozca el lenguaje:

$$L = \{ 0^n 1^n / n \geq 1 \}$$

$\delta(q_0,(0,B)) = (q_0,(0,D),(0,D))$
$\delta(q_0,(1,B)) = (q_1,(1,S),(B,I))$
$\delta(q_1,(1,0)) = (q_1,(1,D),(0,I))$
$\delta(q_1,(B,B)) = (q_A,(B,S),(B,S))$
Las transiciones que faltan van todas a q_R

Notitas:

Sea M una máquina de Turing del modelo "D-I-S" (Derecha, Izquierda, Estático). ¿Existe una máquina de Turing M' equivalente a M que comience con el cabezal apuntando a cualquier celda de la cinta? Note que M' puede apuntar a una celda no ocupada por el input. ¿Qué puede decir al respecto si se sabe que λ ∉ L(M)? Justifique su respuesta.

Si λ pertenece a L(M), entonces cuando M' comienza su ejecución en cualquier celda de la cinta, no puede determinar de inmediato si la cadena de entrada es vacía o no. Esto puede llevar a que M' quede en un bucle infinito si se tratara de que la entrada fuera λ, lo que implicaría que M' estaría rechazando a λ mientras que M la estaría aceptando, lo que hace que M y M' no sean equivalentes en este caso.

Por otro lado, si λ no pertenece a L(M), sí es posible crear una máquina de Turing M' equivalente a M. En este caso, podemos encontrar la cadena original (si es que no caemos dentro) haciendo un movimiento de derecha a izquierda a través de la cinta escribiendo una cadena especial que sirva como marcador para determinar las celdas por las que ya hemos pasado y las que no. Luego, borramos la cadena generada durante el proceso de recorrido y posicionamos el cabezal en la posición inicial de la cadena para que comience la ejecución. A partir de esta instancia, M' utiliza las transiciones originales de M para procesar la entrada y simular su comportamiento.

Practica 4

(odio esta práctica)

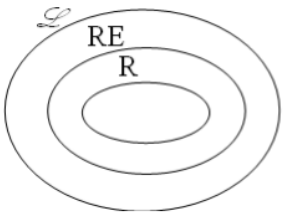
Recursivamente Enumerable

Un lenguaje lo es si y solo si existe una MT M que lo acepte

Recursivo

Existe una MT M tal que L = L(M) y M siempre se detiene para todo input de Σ*

Recordar: Dado un alfabeto Σ, denotamos con Σ* al conjunto de todas las cadenas formadas por símbolos de Σ. $\mathscr{L}$ es el conjunto de todos los lenguajes definidos sobre el alfabeto Σ, es decir $\mathscr{L} = \rho(\Sigma^*)$, es decir el conjunto de todos los subconjuntos posibles de Σ*. Se tiene la siguiente situación:



$$\underbrace{R \subseteq RE \subseteq \mathscr{L}}_{\text{por las definiciones}}$$

1

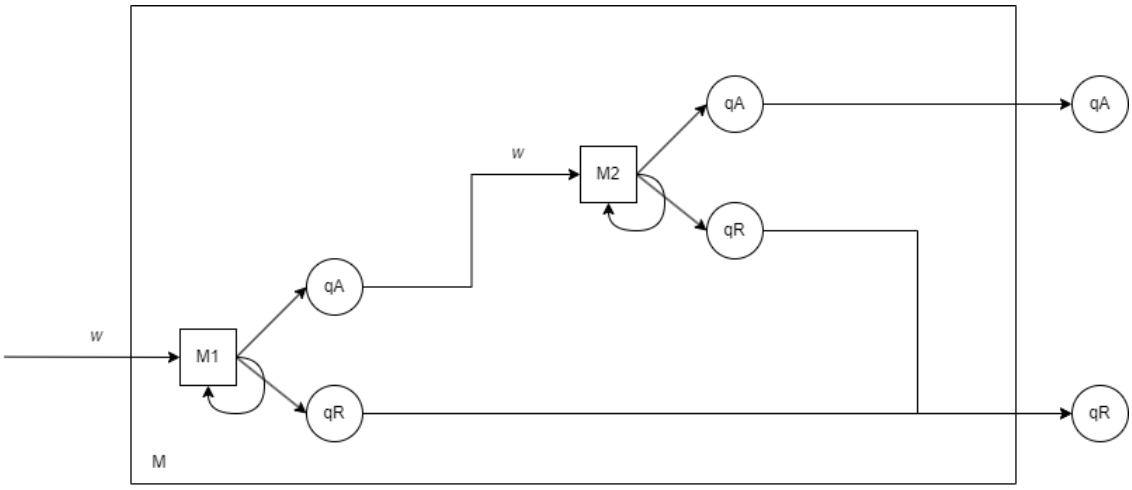
Si un lenguaje es finito → Esta en R

1) Sea L un lenguaje definido sobre Σ. Demostrar que:
   a) $L^c \notin R \Rightarrow L \notin R$

   Si $L^c \notin R \Rightarrow L \notin$ CO-R (definición CO-R)

   $\Rightarrow L \notin R$ (En el teorema 1 y 2 de la teoría se llega a que R = CO-R)

   b) $(L1 \in RE)$ AND $(L2 \in RE) \Rightarrow L1 \cap L2 \in RE$



- L1 = L(M1)
- L2 = L(M2)
- L = L(M)

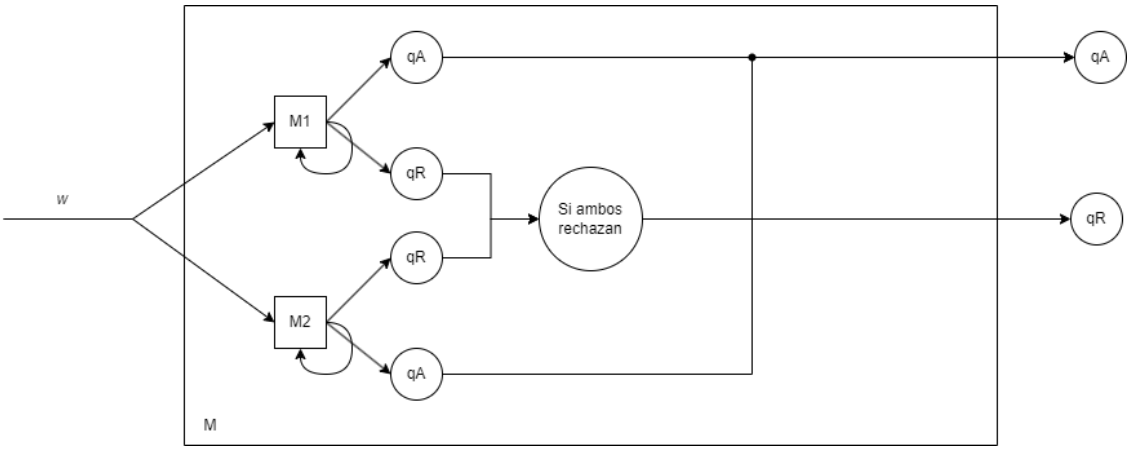
  - L = L1 ∩ L2

  M reconoce solo los inputs que son reconocidos por M1 y M2, es decir, reconoce los inputs que son reconocidos por ambas maquinas (la intersección). Si se trata de un input que rechaza uno pero el otro no, M no lo va a aceptar. Si se trata de un input ambas maquinas, M lo va a rechazar. Por lo tanto L = L1 ∩ L2

  - L ∈ RE

  Como L = L(M), existe una máquina que lo acepta.
  Si M1 o M2 se quedan loopeando M se va a quedar loopeando.

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



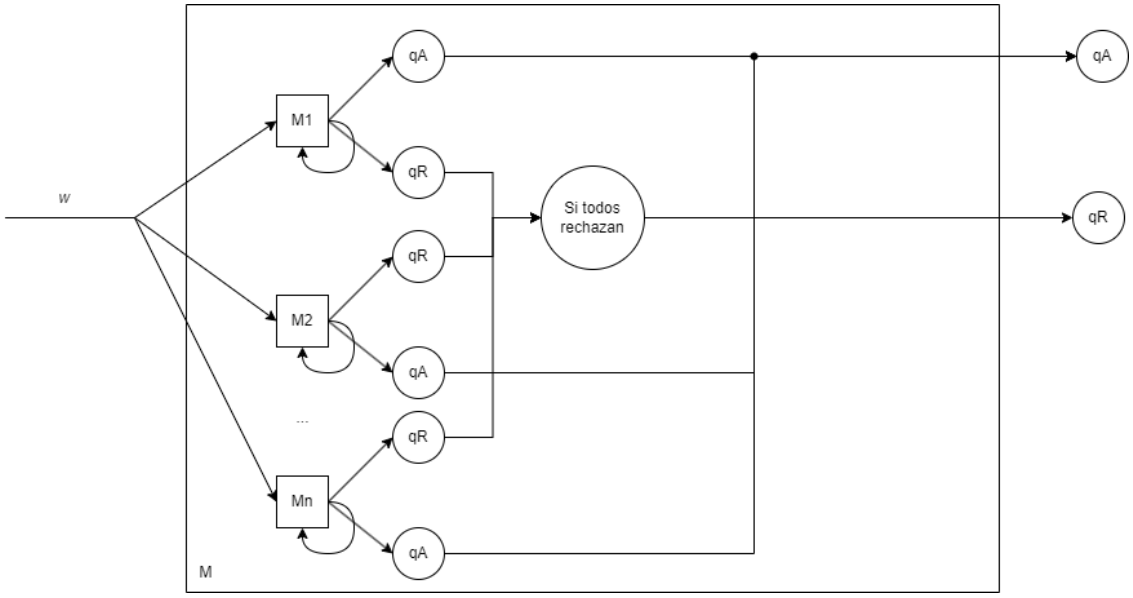
- L1 = L(M1)
- L2 = L(M2)
- L = L(M)

  - L = L1 ∪ L2

  M reconoce los inputs que son reconocidos por M1 o M2, es decir, reconoce los inputs que son reconocidos por alguna de las dos máquinas (la unión).
  M va a simular la ejecución de M1 y M2 de forma alternada. Para ello, se van a simular i pasos de M1 y M2, de forma secuencial repitiendo este proceso para incrementos sucesivos de i.
  Si alguna de las máquinas (M1 o M2) acepta el input, entonces M también acepta el input. Si una máquina rechaza el input, se va a esperar a que la otra también lo rechace. Si ambas máquinas rechazan el input, M lo rechaza.
  - L ∈ RE

  Como L = L(M), existe una máquina que lo acepta.
  Si M1 o M2 se quedan loopeando M se va a quedar loopeando.

  d) La unión de un número finito de lenguajes recursivamente enumerables es un lenguaje recursivamente enumerable.



- L1 = L(M1)

- L2 = L(M2)
- Ln = L(Mn)
- L = L(M)

  - $L = L1 \cup L2 \cup \ldots \cup Ln$

    M reconoce los inputs que son reconocidos por alguna de todas las máquinas (la unión).
    M va a simular la ejecución de M1, M2, … Mn de forma alternada. Para ello, se van a simular i pasos de M1, M2, … Mn de forma secuencial repitiendo este proceso para incrementos sucesivos de i.
    Si alguna de las máquinas acepta el input, entonces M también acepta el input.
    Si una máquina rechaza el input, se va a esperar a que TODAS las otras también lo rechacen.

  - $L \in RE$

    Como L = L(M), existe una máquina que lo acepta.
    Si alguna de las maquinas se queda loopeando M se va a quedar loopeando.

Definiciones:

Un lenguaje L pertenece a CO-R si y solo si L^c pertenece a R → $L^c = \Sigma^* - L$

Un lenguaje L pertenece a CO-RE si y solo si L^c pertenece a RE

Por definición $R \subseteq RE \subseteq L$

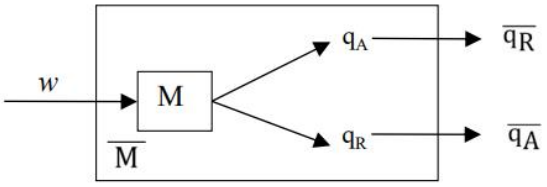
Teoremas:

1. $R \subseteq CO\text{-}R$

Demostración. Hay que demostrar que si $L \in R \Rightarrow L \in$ Co-R, es decir que si $L \in R \Rightarrow \overline{L} \in R$.

Sea $L \in R \Rightarrow$ Existe una MT $M = <Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R>$ tq $L = L(M)$ y M se detiene en algún momento para toda entrada.

Se construye $\overline{M} = <Q, \Sigma, \Gamma, \delta, q_0, \overline{q_A}, \overline{q_R}>$ con $\overline{q_A} = q_R$ y $\overline{q_R} = q_A$



Hay que probar que $L(\overline{M}) = \overline{L}$ y además que $\overline{M}$ se detiene en algún momento para toda entrada.

1) Sea $w \in L(\overline{M}) \Leftrightarrow q_0 w \vdash^+_{\overline{M}} \alpha_1 \overline{q_A} \alpha_2 \Leftrightarrow q_0 w \vdash^+_M \alpha_1 q_R \alpha_2 \Leftrightarrow w \notin L(M) \Leftrightarrow w \notin L \Leftrightarrow w \in \overline{L}$

def.L($\overline{M}$). Constr. def.L(M) Hip. def. $\overline{L}$

Por lo tanto, $L(\overline{M}) = \overline{L}$

2) ¿$\overline{M}$ se detiene para toda entrada?: Sí, por construcción $\overline{M}$ se detiene cuando M se detiene y por hipótesis M se detiene para toda entrada.

De 1) y 2) $L \in R \Rightarrow L \in$ Co-R

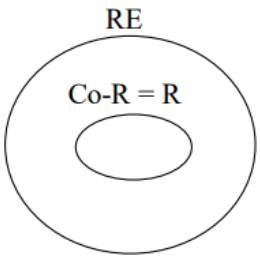
2. CO-R $\subseteq$ R

Sea $L \in$ Co-R $\Rightarrow \overline{L} \in R \Rightarrow \overline{L} \in$ Co-R $\Rightarrow \overline{\overline{L}} \in R \Rightarrow L \in R$

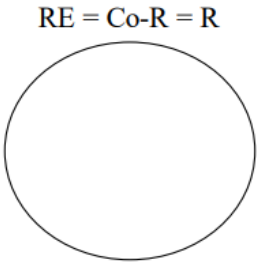
def.Co-R teor. ant. def. Co-R prop. de compl.

Por lo tanto, Co-R $\subseteq$ R

Corolario: de los dos teoremas anteriores surge que R = Co-R



O bien



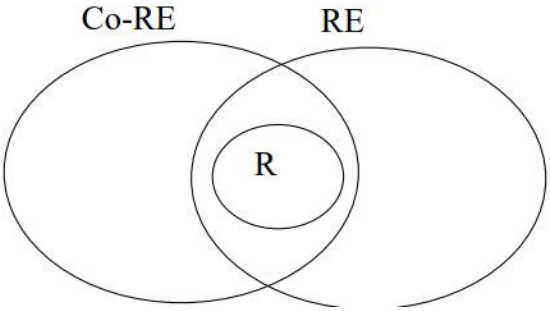
3. R $\subseteq$ CO-RE

Dem.: Sea $L \in R \Rightarrow \overline{L} \in R \Rightarrow \overline{L} \in$ RE $\Rightarrow L \in$ Co-RE

teor. 1 (def.R y RE) def. Co-RE

Por lo tanto, R $\subseteq$ Co-RE.

Además, como por definición R $\subseteq$ RE se tiene que R $\subseteq$ (RE $\cap$ Co-RE)



4. (RE $\cap$ CO-RE) $\subseteq$ R

Sea $L \in (RE \cap Co\text{-}RE)$

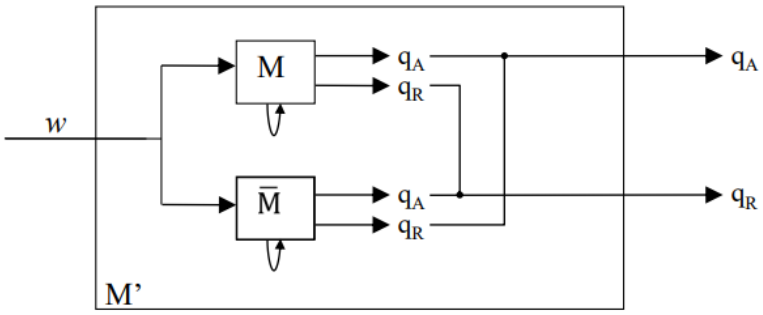
$\Rightarrow L \in RE \wedge \overline{L} \in RE$ (por def. de $\cap$ y de Co-RE)

$\Rightarrow$ existen M y $\overline{M}$, dos MT tq $L = L(M)$ y $\overline{L} = L(\overline{M})$

Hay que construir una MT M' que reconozca L y que se detenga siempre.

$\forall w \in \Sigma^*$ o bien $\overline{M}$ para en q_A o bien M para en q_A (no puede darse nunca el caso que ambas "loopeen").

Por lo tanto hay que construir M' simulando en paralelo M y $\overline{M}$, si M para en q_A o $\overline{M}$ para en $q_R \Rightarrow$ M' para en q_A y si M para en q_R o $\overline{M}$ en $q_A \Rightarrow$ M' para en q_R.



¿Cómo simular dos máquinas en paralelo? No importa la eficiencia, en una máquina de 4 cintas:

1) Escribir el número 1 en la cinta 4 (sea *i* ese valor).

2) Copiar w a las cintas 2 y 3.

3) Simular a lo sumo *i* pasos de M en la cinta 2 y a lo sumo *i* pasos de $\overline{M}$ en la cinta 3. si M para en q_A o $\overline{M}$ para en $q_R \Rightarrow$ M' para en q_A y si M para en q_R o $\overline{M}$ para en $q_A \Rightarrow$ M' para en q_R.

4) Borrar las cintas 2 y 3. Incrementar *i* en la cinta 4 y volver al punto 2.

Demostrar como ejercicio que L = L(M') y que M' se detiene siempre.

Pregunta: ¿Cómo se pueden simular *i* pasos?

Corolario: (RE ∩ Co-RE) = R (por los teoremas 3 y 4)

Orden canónico para Σ* → se listan todas las palabras en orden según su tamaño con las palabras del mismo tamaño en orden lexicográfico

Ej: Σ = {0, 1}, el orden canónico es:

λ	0	1	00	01	10	11	000	001	010	011	100	101	110	111	0000	0001	…
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	

Obsérvese que si *w* es un string de {0,1}*, la posición *i* que ocupa en el orden canónico se escribe en binario como 1*w*. Decimos entonces que *w* es el *i*-ésimo string y por ello lo denotamos w_i

Por ejemplo el string λ ocupa la posición 1 (1λ) el string 01 ocupa la posición 5 (101), el string 0000 la posición 16 (10000)

Se puede codificar una MT como un string de un alfabeto de 2 símbolos:

$Q = \{q_0, q_1, \ldots, q_K\}$ $\Sigma = \{a_1, a_2, \ldots, a_L\}$ $\Gamma = \{B, a_1, a_2, \ldots, a_L, a_{L+1}, \ldots, a_n\}$

Se puede codificar M con un alfabeto binario {0, 1} de la sgte. forma:

<u>Estados</u>: $q_A = 1$ $q_R = 11$ $q_0 = 111$ $q_1 = 1111$ … $q_i = 1^{(i+3)}$

<u>Símbolos</u>: $B = 1$, $a_1 = 11$ $a_2 = 111$ … $a_i = 1^{(i+1)}$

<u>Movimiento del Cabezal</u>: $D = 1$ $I = 11$ $S = 111$

<u>Función de Transición</u>: cada transición se codifica como una quíntupla de elementos separados por un símbolo 0. Ej.:

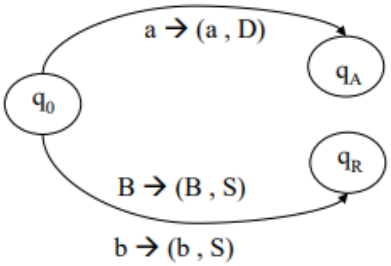
$\delta(q_0, a_2) = (q_1, a_4, D)$ se codifica como

$(q_0, a_2, q_1, a_4, D) = (1110111011110111101)$

q_0, a_2, q_1, a_4, D

M se codifica como una sucesión de quíntuplas separadas por 00.

Ej.: $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$ $Q = \{q_0\}$ $\Sigma = \{a, b\}$ $\Gamma = \{B, a, b\}$



$d(q_0, a) = (q_A, a, D)$
$d(q_0, b) = (q_R, b, S)$
$d(q_0, B) = (q_R, B, S)$

Ej.: $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$ $Q = \{q_0\}$ $\Sigma = \{a, b\}$ $\Gamma = \{B, a, b\}$

$q_A = 1$	$q_R = 11$	$q_0 = 111$
$B = 1$	$a = 11$	$b = 111$
$D = 1$	$I = 11$	$S = 111$

$\langle M \rangle = 1110110101101001110111011011101110011101011010111$

$\delta(q_0, a) = (q_A, a, D)$ $\delta(q_0, b) = (q_R, b, S)$ $\delta(q_0, B) = (q_R, b, S)$

Note que existen otros posibles códigos para M. Las tres transiciones pueden listarse en distinto orden (3! formas distintas) por lo tanto hay 6 códigos distintos para la misma M

23

<M> → Codificacion de Maquina de Turing

L<M> ∈ R? Si porque existe una Maquina de Turing que realiza el chequeo sintactico y siempre se detiene.

Se denomina i-ésima MT y se denota Mi a la MT cuyo código es wi (i-ésimo string binario en orden canónico), es decir <Mi> = wi.

Si wi no es un codigo valido de maquina de turing codifica a Mi siendo Mi una MT que se detiene inmediatamente rechazando cualquier entrada, por lo tanto L(Mi)={}. Así, todo string wi de {0,1}* se corresponde con, o codifica, una MT Mi

LENGUAJE DIAGONAL (IMPORTANTE)

Es un lenguaje no recursivamente enumerable.

LD = {wi ∈ Σ* / wi ∉ L(Mi)}

siendo Σ ={0,1}; wi el i-ésimo string en orden canónico de S * y Mi la i-ésima MT.

Son codificaciones de MT que rechazan su propia codificación.

$$L_D = \{w_i \in \Sigma^* \,/\, w_i \notin \mathrm{L}(\mathrm{M}_i)\}$$

Teorema: $L_D \notin$ RE

Dem.: Por reducción al absurdo, se asume que $L_D \in$ RE $\Rightarrow$ existirá una MT M que lo acepta y cuya codificación está en alguna posición del orden canónico, es decir que existe algún natural k para el cual $\mathrm{M} = \mathrm{M}_k$ y $L_D = \mathrm{L}(\mathrm{M}_k)$.

Considerando w_k el k-ésimo string de Σ^*, hay dos posibilidades, o bien $w_k \in L_D$ o bien $w_k \notin L_D$

a) $w_k \in L_D \Rightarrow w_k \notin \mathrm{L}(\mathrm{M}_k) \Rightarrow$ (por def. L_D)

$\Rightarrow w_k \notin L_D$ (porque $\mathrm{L}(\mathrm{M}_k) = L_D$)

(contradicción)

b) $w_k \notin L_D \Rightarrow w_k \in \mathrm{L}(\mathrm{M}_k)$ (por def. L_D)

$\Rightarrow w_k \in L_D$ (porque $\mathrm{L}(\mathrm{M}_k) = L_D$)

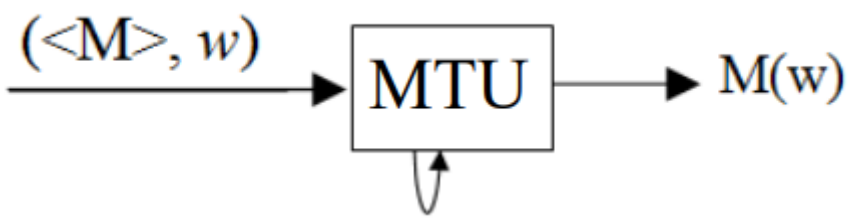
(contradicción)

En ambos casos se llega a una contradicción y por ende no puede existir una M_k que reconozca al lenguaje L_D, por lo tanto L_D no puede ser recursivamente enumerable.

27

Por lo tanto $L_D \notin$ RE.

MTU → Maquina de Turing Universal → es una MT que recibe como entrada la codificación de otra MT M y una entrada w. La MTU ejecuta la MT M sobre la entrada w.



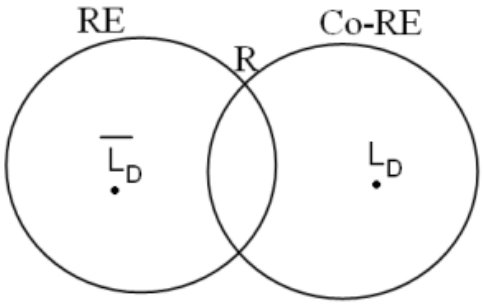
Se tiene una versión reconocedora.

Nota1: puede probarse que L_D pertenece a Co-RE, pues fácilmente se verifica que:

$$\overline{L_D} \in \text{RE} \quad (\overline{L_D} = \{w_i \in \Sigma^* \,/\, w_i \in \text{L}(\text{M}_i)\})$$

Para ello se construye una MT que utilizando una MTU acepta $\overline{L_D}$ ejecutando el código de M_i sobre w_i aceptando sii M_i acepta w_i. Recuérdese que el código $<\text{M}_i>$ de la MT M_i se obtiene fácilmente pues $<\text{M}_i> = w_i$

Nota 2: Además $\overline{L_D}$ no puede estar en R, ya que esto implicaría que L_D también esté en R (y ya sabemos que L_D no pertenece a RE). Por lo tanto $\overline{L_D} \in$ RE - R.



El lenguaje universal esta formado por las codificaciones de maquina de turing que aceptan un input.

Def.: se define L_u , el lenguaje universal, como:

$$L_u = \{(<M>, w) \text{ / M acepta } w\}$$

Pregunta: ¿$L_u \in$ RE?

Rta.: claramente sí. Se puede construir M_u de la siguiente manera:

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

2) En caso contrario separar <M> de w

3) Si <M> es un código inválido parar en q_R

4) Simular M sobre w. Si M para en $q_A \Rightarrow M_u$ para en q_A. Si M para en $q_R \Rightarrow M_u$ para en q_R. Si M loopea $\Rightarrow M_u$ también loopea.

Claramente $L_u = L(M_u)$, por lo tanto $L_u \in$ RE 34

$L_u = \{(<M>, w) \text{ / M acepta } w\}$ entonces :

$(<M>, w) \in \overline{L_u}$ sii (<M>, w) no es un par válido o M rechaza w

Teorema: $\overline{L_u} \notin$ RE

Dem.: se demuestra que si $\overline{L_u} \in$ RE $\Rightarrow L_D \in$ RE, lo cual es absurdo pues ya se vio que $L_D \notin$ RE.

Si $\overline{L_u} \in$ RE $\Rightarrow \exists$ MT $\overline{M_u}$ que acepta $\overline{L_u}$, es decir que dada una entrada (<M>, w) puede identificar que M rechaza w

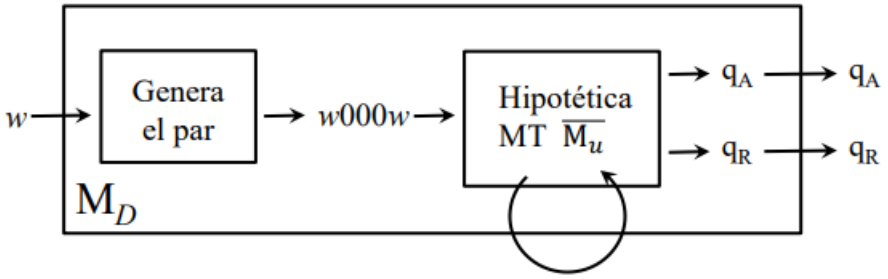
Considerando que existe tal $\overline{M_u}$ se construye la MT M_D que acepta L_D de la siguiente manera:

1) M_D a partir de la entrada w genera el par $w000w$ (recordar que $<M_i> = w_i$)

2) M_D simula $\overline{M_u}$ sobre $w000w$. M_D acepta w sii $\overline{M_u}$ acepta $w000w$

L_u = {(<M>, w) / M acepta w} entonces :

(<M>, w) $\in \overline{L_u}$ sii (<M>, w) no es un par válido o M rechaza w



Si w es w_i en nuestra numeración se tiene que:

M_D acepta w_i $\Leftrightarrow$ $\overline{M_u}$ acepta <M_i>000w_i $\Leftrightarrow$ M_i rechaza w_i $\Leftrightarrow$ $w_i \in L_D$

De esta forma $L(M_D) = L_D$ lo que significa que $L_D \in$ RE

Por lo tanto demostramos que si $\overline{L_u} \in$ RE $\Rightarrow L_D \in$ RE,

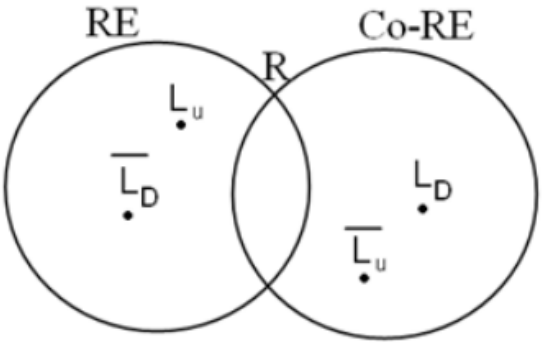
Por contrarrecíproca se tiene que $L_D \notin$ RE $\Rightarrow \overline{L_u} \notin$ RE

y como ya se conoce que $L_D \notin$ RE se tiene que $\boldsymbol{\overline{L_u} \notin}$ **RE**

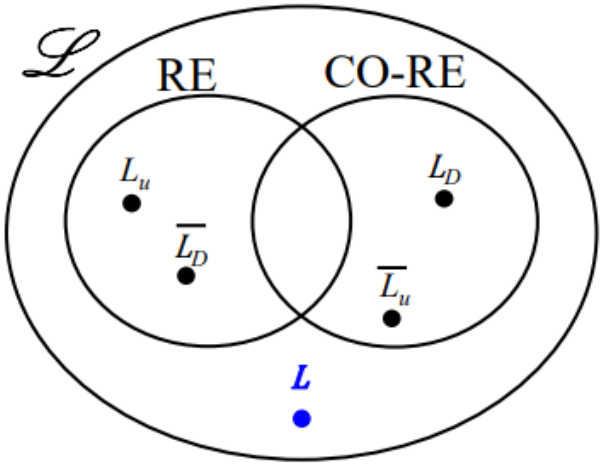
Corolario: $L_u \in$ (RE – R).

Inmediato pues ya sabemos que L_u está en RE y que L_u no puede estar en R pues ello implicaría que $\overline{L_u}$ también estuviese en R lo que sería un absurdo pues acabamos de demostrar que $\overline{L_u} \notin$ RE

Hasta acá se tiene entonces la siguiente situación:



Lenguaje del conjunto $\mathscr{L} - (\text{RE} \cup \text{CO-RE})$



$$L = \{1w \,/\, w \in L_D\} \cup \{0w \,/\, w \notin L_D\}$$

$$L \in \mathscr{L} - (\text{RE} \cup \text{CO-RE})$$

39

ENTONCES RESUMIENTO

Lenguaje Diagonal → A este lenguaje pertenecen las codificaciones de MT que rechazan su propia codificación

Pertenece a CO-RE – RE. Su complemento pertenece a RE – CO-RE. Como RE ∩ CO-RE = R, pertenece a CO-RE - R

Lenguaje Universal → A este lenguaje pertenecen las codificaciones de MT que aceptan un input.

Pertenece a RE – CO-RE. Su complemento pertenece a CO-RE – RE. Como RE ∩ CO-RE = R , pertenece a RE – R

https://www.youtube.com/watch?v=LU9C-mlDLJY

TEOREMAS (sin demostraciones están más arriba)

1. R ⊆ CO-R
2. CO-R ⊆ R
3. R ⊆ CO-RE
4. (RE ∩ CO-RE) ⊆ R

ENTONCES CO-RE ∩ RE = R

Recordar para futuras demostraciones:

- Resta de un infinito incontable con uno contable da incontable

- La unión de dos conjuntos contables da uno contable

  (esto es de la practica 1)

  Mostrar que la unión de 2 conjuntos contables es contable

  $|(N - N^+) \cup N^+| = |N|$
  $|N| = |N|$

  Muestre que, si X es un conjunto incontable e Y es un conjunto contable, entonces X-Y debe ser incontable

  X = R
  Y = {y/y ∈ N, y es un numero impar}

  Probar que:

  $|X| = |X – Y|$

  Para ello:

  18.1) $|X – Y| <= |X|$
  18.2) $|X| <= |X – Y|$

  Para probar 18.1) es sencillo mediante la función de identidad. Para probar 18.2) podemos utilizar la función inyectiva dbl: R → X - Y ,dbl(r) = 2r

  Así hemos demostrado que si a un conjunto incontable (todos los reales), le resto un conjunto contable (los números impares), obtengo un conjunto incontable.

- Que una MT acepte un lenguaje recursivo no quiere decir que esa maquina se detenga, si no que existe una máquina de Turing (puede ser tranquilamente otra) que acepta ese lenguaje y siempre se detiene, pero no en particular esta.

  Demostraciones de la practica que vale la pena recordar:

  2) Sean Σ ={a,b} y $\mathcal{L}$ el conjunto de todos los lenguajes definidos sobre Σ. Diga si las siguientes afirmaciones son verdaderas o falsas:

     a) $\mathcal{L} – R = \emptyset$

        Falso

        $L_D \in \mathcal{L}$ pero no a R, por lo que si se hace $\mathcal{L} – R$ por lo menos L_D seguirá estando en $\mathcal{L}$

b) $\Sigma^* \in R$

Verdadera

Existe una MT tal que acepta el lenguaje $\Sigma *$ (q0w $\vdash$ qA)

c) $ab \in \Sigma *$

Verdadera

$\Sigma *$ = (λ,a,b,aa,ab,ba,bb,…)

d) $RE - R \neq \emptyset$

Verdadero

Por absurdo: $RE - R = \emptyset$

$Lu \in (RE - R)$

$Lu \in \emptyset$ (ABSURDO)

$\therefore RE - R \neq \emptyset$

e) $\emptyset \in RE$

Verdadero

Existe una MT que acepta el lenguaje $\emptyset$ tal que (q0w $\vdash$ qR) $\therefore \emptyset \in R$ y por definición $\emptyset \in RE$

f) $CO\text{-}R \subset CO\text{-}RE$

Verdadero

$L \in CO\text{-}R \Rightarrow$ (def. Co-R)

$\bar{L} \in R \Rightarrow$ (def. R y RE)

$\bar{L} \in RE \Rightarrow$ (def. Co-Re)

$L \in CO\text{-}RE$

g) $\{\lambda\} \in (\mathcal{L} - CO\text{-}RE)$

Falso

Existe una MT tal que acepte el lenguaje {λ} y sea recursivo (q0w $\vdash$ qA, w =λ) o (q0w $\vdash$ qR))

$\therefore \{\lambda\} \in R \Rightarrow \{\lambda\} \in CO\text{-}RE$ (Teorema 3)

$\therefore \{\lambda\} \notin (\mathcal{L} - CO\text{-}RE)$

h) $CO\text{-}RE = RE$

Falso

Por absurdo: CO-RE - RE = $\emptyset$

$L_D \in$ (CO-RE - RE)

$L_D \in \emptyset$ (ABSURDO)

$\therefore$ CO-RE $\neq$ RE

i) $a \in R$

Falso

R = {$\emptyset$, {λ}, {a}, {b}, {aa}, {ab}, …) *a $\notin$ R.*

*"a" no es un lenguaje. El conjunto R se compone de lenguajes (conjuntos de cadenas), no de cadenas individuales (*a es una cadena y {a} es el lenguaje que contiene la cadena a)

La diferencia entre "a $\notin$ R" y "{a} $\in$ R" se debe a la forma en que se definen los lenguajes y las cadenas en la teoría de la computabilidad.

<u>a $\notin$ R (a no pertenece a R):</u>

En el contexto de la teoría de la computabilidad, el conjunto de lenguajes recursivos (R) está formado por lenguajes que son decidibles por una máquina de Turing, es decir, hay una máquina de Turing que puede aceptar o rechazar cada cadena de entrada en un tiempo finito. Un lenguaje se considera recursivo si hay una MT que siempre se detiene y decide si una cadena dada está en el lenguaje o no.

La notación "a $\notin$ R" significa que la cadena "a" no pertenece a ningún lenguaje recursivo. Esto implica que no existe una máquina de Turing que pueda tomar la cadena "a" como entrada y siempre detenerse en un tiempo finito para decidir si "a" está en el lenguaje o no. En otras palabras, "a" es una cadena que no es decidible por ninguna máquina de Turing.

<u>{a} $\in$ R ({a} pertenece a R):</u>

Por otro lado, cuando tienes un conjunto que contiene una sola cadena, como "{a}", esta notación se refiere al lenguaje que contiene solo esa cadena. En este caso, el lenguaje es {a}. El hecho de que "{a} $\in$ R" significa que el lenguaje que contiene solo la cadena "a" es un lenguaje recursivo. Esto implica que existe una máquina de Turing que puede decidir en tiempo finito si una cadena dada es igual a "a" o no.

j) RE $\cup$ R = $\mathcal{L}$

Falso

RE $\cup$ R = RE

$L_D \in \mathcal{L}$ pero $L_D \notin$ RE

$\therefore$ RE $\cup$ R $\neq \mathcal{L}$

k) ($\mathcal{L}$ – RE) = CO-RE

Falso

Existe L = {1w / w $\in L_D$ } $\cup$ {0w / w $\notin L_D$ } que $\notin$ (RE $\cup$ CO-RE)

Existe un lenguaje en $\mathcal{L}$ - RE que no está en CO-RE

$\therefore$ ($\mathcal{L}$ – RE) ≠ CO-RE, ($\mathcal{L}$ – RE) = L $\cup$ CO-RE

l) {a} ∈ RE

Verdadero

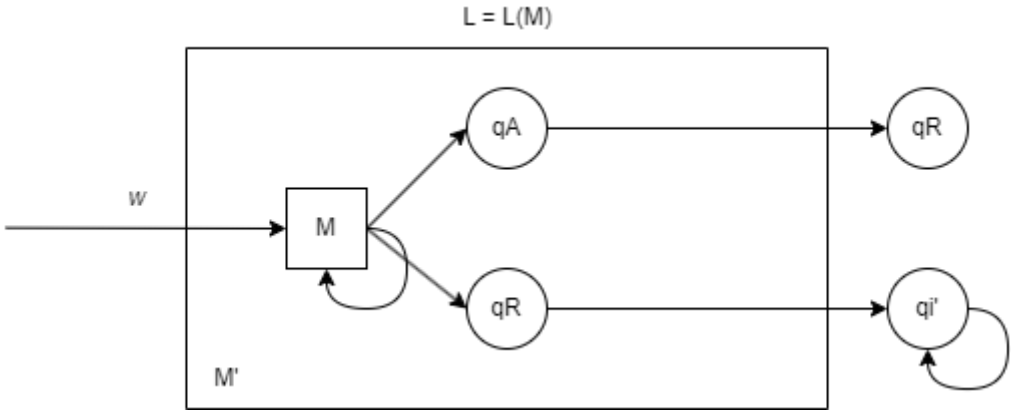
{a} ∈ R ⇒ (def. R y RE)

{a} ∈ RE

3) Si L∈(RE - R)

a) ¿Existirá alguna máquina de Turing que rechace parando en qR si su entrada está en L y rechace loopeando si su entrada no está en L?

Si, sabiendo que tenemos una maquina que reconoce L y que a veces rechaza loopeando, podemos construir una maquina tomando la anterior y todas las δ que terminan en estado qR las redirijo a un loopeo y todas las que terminan en qA las redirijo a qR



b) ¿Existirá alguna máquina de Turing que rechace loopeando si su entrada está en L y rechace parando en qR si su entrada no está en L?

No, no se puede crear una máquina que, a partir de la maquina original, rechace loopeando si su entrada está en L y rechace parando en qR si su entrada no está en L, porque la maquina original cuando la entrada no esta en L se puede quedar loopeando, no siempre para en qR. Si rechaza loopeando no hay forma de identificar que la maquina entro en un loop como para rechazarla parando en qR

Demostración por absurdo

- M (máquina que reconoce a L)

- Si el string pertenece a L se detiene en qA

- Mb (máquina del inciso b que rechaza loopeando si su entrada está en L y rechace parando en qR si su entrada no está en L)
    - Si el string no pertenece a L se detiene en qR

Podría construir una nueva maquina si yo recibo un string lo hago pasar por estas dos máquinas paralelamente. O la primer maquina se va a detener en qA o la segunda se va a detener en qR. Para todo string alguna de las dos máquinas se va a detener.

Si tenemos garantizado que alguna de las maquinas se va a detener, podemos construir una nueva maquina que usa las dos máquinas mencionadas, que seguro se detiene y que me puede decir si un string pertenece o no a L

Si podemos construir una maquina así, entonces tengo una maquina que reconoce L y siempre se detiene, eso significa que $L \in R \rightarrow$ absurdo, partimos de que $L \in (RE - R)$ y llegamos a la conclusión de que $L \in R$.

c) De existir, que lenguaje reconocería esta máquina de Turing.

Reconocería el $\emptyset$ porque rechazaría los que L(M) acepta y también rechazaría los que L(M) rechaza.

4) Sea L ={w | Existe alguna Máquina de Turing M que acepta w} ¿$L \in R$? Justifique.

Para que una palabra quede afuera de L, tendría que existir una máquina de Turing que no acepte esa palabra. Todo los Strings de mi alfabeto van a tener al menos una máquina que lo acepte. No hay un String que va a quedar por fuera de L
Para todo w, existe alguna Máquina de Turing M que acepta w.

- $L = \Sigma *$.

$\therefore L \in R$, porque puedo hacer una Maquina de Turing que me lee el primer símbolo carácter y automáticamente pare en qA, esta maquina reconoce el

lenguaje $\Sigma *$ y como lo reconoce y siempre se detiene podemos decir que $\Sigma * \in$ R.

5) Conteste y justifique:

a) ¿$\mathcal{L}$ es un conjunto infinito contable?

No, es incontable ya que $\mathcal{L}=\rho(\Sigma*)$, y ya está demostrado que $\rho(\Sigma*)$ es incontable.

b) ¿RE es un conjunto infinito contable?

RE es infinito contable ya que cada lenguaje de este conjunto puede asociarse con una maquina de Turing que lo reconozca, y el conjunto de máquinas de Turing es contable. Aunque un lenguaje puede ser aceptado por varias maquinas de Turing, aun podemos contar ordenadamente estas máquinas, demostrando que el conjunto de lenguajes recursivamente enumerables es infinito contable.

c) ¿$\mathcal{L}$ – RE es un conjunto infinito contable?

- La unión de dos conjuntos contables da otro conjunto contable

($\mathcal{L}$ – RE) U RE es contable → Absurdo
($\mathcal{L}$ – RE) U RE = $\mathcal{L}$ y este es incontable, para que la unión sea incontable (que es lo que debería ser) lo que esta a la izquierda tiene que ser incontable (sabemos que RE si o si es contable).

d) Existe algún lenguaje L $\in \mathcal{L}$, tal que L sea infinito no contable

Todo lenguaje es subconjunto de $\Sigma*$, dado que ya sabemos que $\Sigma*$ es infinito contable, un subconjunto de el va a ser a lo sumo infinito contable, por lo que no existe algún lenguaje L que $\in \mathcal{L}$ tal que sea infinito no contable.

Demostración:
Sabemos que la unión de dos conjuntos contables da otro conjunto contable. También sabemos que $\Sigma*$ es infinito contable.
Supongamos que existe un L infinito no contable, esto llevaría a que

L U L^C sea incontable (ya que en la unión, si al menos uno es incontable, la unión es incontable).

L U L^C = $\Sigma*$ ∴ $\Sigma*$ es incontable → Absurdo, ya que partimos sabiendo que $\Sigma*$ es infinito contable

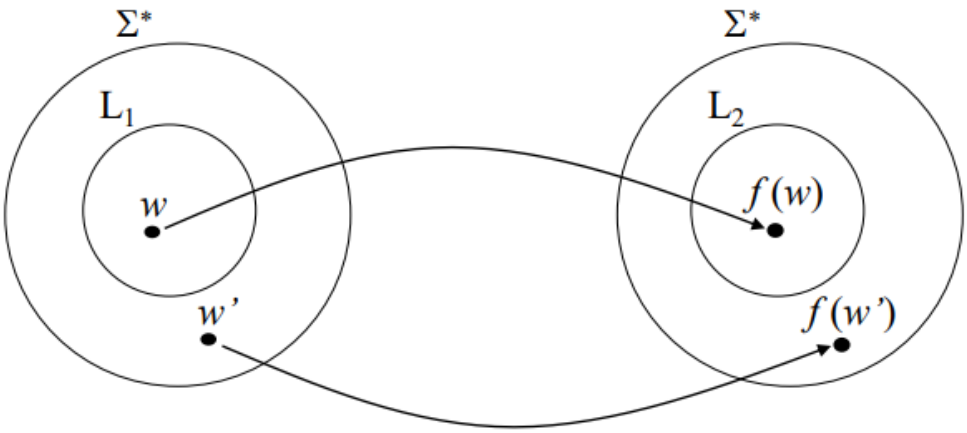
Practica 5

Reducibilidad

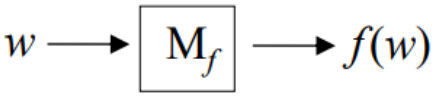
Reducibilidad

- **Def.**: Sean $L_1, L_2 \subseteq \Sigma^*$ se dirá que L_1 se reduce a L_2 (L_1 α L_2) si existe una función total computable (o recursiva) $f: \Sigma^* \rightarrow \Sigma^*$ tal que

$$\forall w \in \Sigma^*, w \in L_1 \Leftrightarrow f(w) \in L_2$$



F es computable si existe una MT que la computa y siempre se detiene (realiza n pasos FINITOS)



M_f nunca loopea. A veces, usaremos la expresión $M_f(w)$ para referirnos a la función computada por la MT M_f

Reducibilidad → transformación de instancias de un problema en instancias de otro problema.

Para demostrar si existe una reducibilidad hay que demostrar:

Puede demostrarse fácilmente que:

1. M_f siempre se detiene
2. $\forall w \in \Sigma^*, w \in L_1 \Leftrightarrow f(w) \in L_2$

Para w ∈ L1 ⇔ Mf (w) ∈ L2 es necesario demostrar que:

- w ∈ L1 ⇒ Mf (w) ∈ L2
- w ∉ L1 ⇒ Mf (w) ∉ L2

si L1 α L2, L1 no puede ser más difícil computacionalmente que L2 porque se puede utilizar L2 para resolver L1

Intuitiva y coloquialmente, la reducción establece una relación de "≤" grado de dificultad computacional entre lenguajes/problemas.

Reducibilidad

Teorema 1: $L_1, L_2 \subseteq \Sigma^*$ y existe la reducción L_1 α L_2, entonces:

$$\text{si } L_2 \in R \Rightarrow L_1 \in R$$

Teorema 2: $L_1, L_2 \subseteq \Sigma^*$ y existe la reducción L_1 α L_2, entonces:

$$L_2 \in RE \Rightarrow L_1 \in RE$$

Corolario: Sean $L_1, L_2 \subseteq \Sigma^*$ y la reducción L_1 α L_2, por las respectivas contrarecíprocas del teorema 1 y del teorema 2, se cumple que:

$$L_1 \notin R \Rightarrow L_2 \notin R$$
$$L_1 \notin RE \Rightarrow L_2 \notin RE$$

Halting Problem

HP = {(<M>, w) tal que M se detiene con input w}

HP ∈ (RE-R) → Esto es así porque HP ∈ RE y existe una reducción de Lu a HP, como Lu ∉ R entonces HP ∉ R

LΣ* = { <M> / L(M) = Σ*} ∉ R ni ∉ RE (existe una reducción de Lu a LΣ* y también existe una reducción Lu^c u α LΣ * y como Lu^c u ∉ RE se tiene que LΣ* ∉ a RE

Ejercicio: Probar que $L_{\Sigma^*} = \{<M> / L(M) = \Sigma^*\}$ no es recursivo.

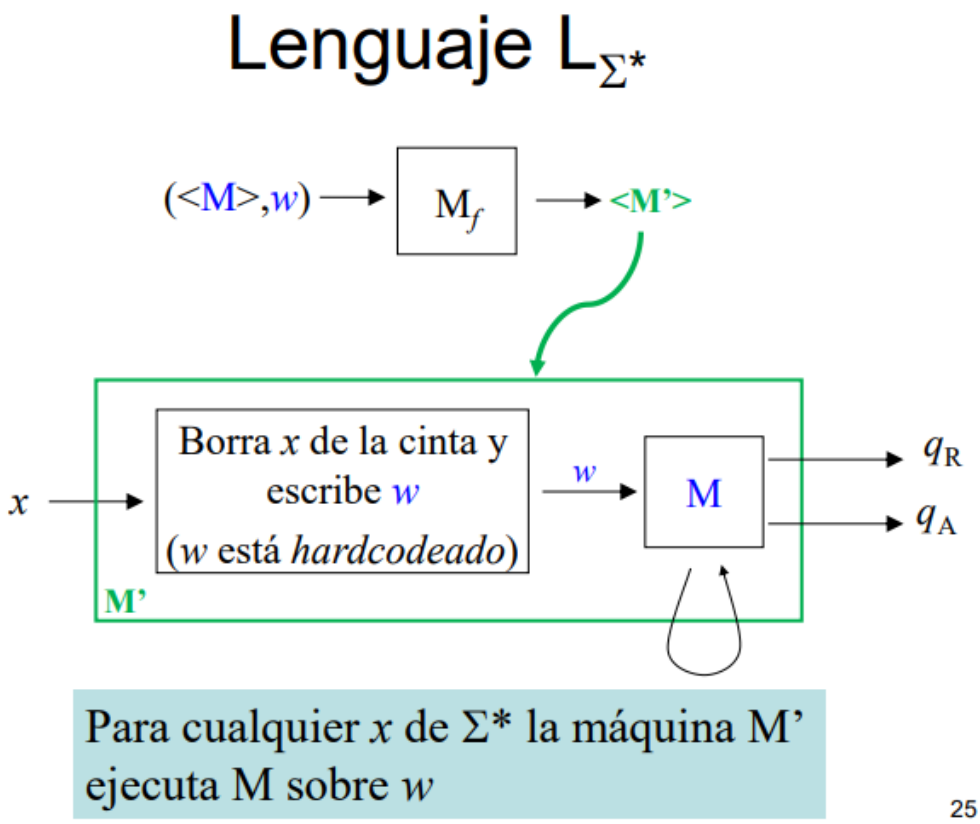
Mostraremos que existe una reducción $L_u \ \alpha \ L_{\Sigma^*}$

Hay que encontrar una función total computable tal que

$$(<M>,w) \in L_u \Leftrightarrow f(<M>,w) \in L_{\Sigma^*}$$

Sea M_f la máquina de Turing que computa la función $f(<M>,w) = <M'>$ y trabaja de la siguiente manera:

Si $(<M>,w)$ no es un par válido o $<M>$ no es un código de MT válido, M_f borra la cinta (deja λ como salida), de lo contrario M_f construye $<M'>$ escribiendo las quíntuplas necesarias para que M' borre la entrada y escriba w en la cinta, posicione el cabezal y simule M sobre w. Así M' para en q_A para cualquier input $\Leftrightarrow$ M acepta w



1) f ((<M>,w)) es computable? Claramente sí, pues M_f se detiene luego de realizar una cantidad finita de acciones.

2) ¿(<M>,w)$\in L_u$ $\Leftrightarrow$ <M'> $\in L_{\Sigma*}$?

a) Sea (<M>,w)$\in L_u$ $\Rightarrow$ M para en q_A con entrada w $\Rightarrow$M' para en q_A con cualquier entrada $\Rightarrow$<M'> $\in L_{\Sigma*}$

b) Sea (<M>,w) $\notin L_u$ $\Rightarrow$

i. Si (<M>,w) no es un par válido o <M> no es un código de MT válido $\Rightarrow$ <M'> = λ $\Rightarrow$ <M'> $\notin L_{\Sigma*}$

ii. M rechaza w $\Rightarrow$ M' rechaza toda entrada $\Rightarrow$<M'> $\notin L_{\Sigma*}$

De a) y b) se tiene que (<M>,w)$\in L_u$ $\Leftrightarrow$ <M'> $\in L_{\Sigma*}$

De 1) y 2) se tiene que L_u α $L_{\Sigma*}$

Por lo tanto $L_{\Sigma*}$ $\notin$ a R (porque L_u $\notin$ a R)

26

L∅ ={<M> / L(M)= ∅}

LEQ = {(<M1>,<M2>) / L(M1)=L(M2)} $\notin$ RE

- Mirar los últimos puntos de la pr 5

NOTAS: Sólo el conjunto vacío es reducible al conjunto vacío, mientras que el conjunto vacío es reducible a todos los conjuntos, excepto Σ∗. El conjunto Σ∗ puede ser reducido a cualquier otro lenguaje menos el vacio.

No se puede reducir NUNCA NUNCA al conjunto vacio o a Σ∗

Practica 6

Notación Asintótica

Asociada al crecimiento de funciones

Una función **t(n)** está **en el orden de f(n)** si existe una constante real positiva **c** y un umbral $\mathbf{u_0}$ tal que $t(n) \leq c.f(n)\ \forall\ n > u_0$

Notación: Conjuntos

N (incluye el 0)

$R^{\geq 0}$: $x \in R$, $x \geq 0$

R^{+}: $x \in R$, $x > 0$

$$O(f(n)) = \{t:N \rightarrow R^+ / \exists\, c \in R^+, n_0 \in N \text{ tq } t(n) \leq c\, f(n), n \geq n_0\}$$

Cota superior

n0 es el umbral donde todas las entradas terminan siendo mayores o iguales que la funcion t(n)

$$\Omega(f(n)) = \{t:N \rightarrow R^+ / \exists\, c \in R^+, n_0 \in N \text{ tq } t(n) \geq c\, f(n), n \geq n_0\}$$

Cota inferior

$$\Theta(f(n)) = \{t:N \rightarrow R^+ / \exists\, c_1, c_2 \in R^+, n_0 \in N \text{ tq } c_1\, f(n) \leq t(n) \leq c_2\, f(n), n \geq n_0\}$$

Acotacion mas precisa (toma tanto O como Ω)

Propiedades

- $g(n) \in \Omega(f(n))$ sii $f(n) \in O(g(n))$ (Regla de Dualidad)
- $g(n) \in \Theta(f(n))$ sii $g(n) \in O(f(n))$ y $g(n) \in \Omega(f(n))$
- $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$

Propiedades

- *Reflexividad* y *Transitividad* de la pertenencia a O(), Ω () y Θ()
- $f(n) \in O(f(n))$
- Si $f(n) \in O(g(n))$ y $g(n) \in O(h(n))$ ==> $f(n) \in O(h(n))$

  - Reflexividad:

$f(n) \in O(f(n))$
$f(n) \in \Theta(f(n))$
$f(n) \in \Omega(f(n))$

- Transitividad:
  Si $f(n) \in O(g(n))$ y $g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$
  Si $f(n) \in \Theta(g(n))$ y $g(n) \in \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n))$
  Si $f(n) \in \Omega(g(n))$ y $g(n) \in \Omega(h(n)) \Rightarrow f(n) \in \Omega(h(n))$
- Simetría:
  $f(n) \in \Theta(g(n)) \Longleftrightarrow g(n) \in \Theta(f(n))$
- Simetría traspuesta:
  $f(n) \in O(g(n)) \Longleftrightarrow g(n) \in \Omega(f(n))$

Regla del umbral

El umbral no es necesario si se trata de funciones positivas

Regla del Umbral: $f, t: N \rightarrow R^+$,
$t(n) \in O(f(n))$ <==> existe $c \in R^+$ tal que
$t(n) \leq c\, f(n)$ para todo natural n

Regla del máximo

$$f, g: N \rightarrow R^+,\ O(f(n) + g(n)) = O(\max(f(n), g(n)))$$

- Vale para Θ
- Vale para suma de cualquier cantidad de funciones
- Si las funciones no son N -> R+ es erroneo.

Regla del limite

1) $\lim_{n \rightarrow inf.} f(n) / g(n) \in R^+ \Longrightarrow f(n) \in O(g(n))$ y $g(n) \in O(f(n))$
2) $\lim_{n \rightarrow inf.} f(n) / g(n) = 0 \Longrightarrow f(n) \in O(g(n))$ y $g(n) \notin O(f(n))$
3) $\lim_{n \rightarrow inf.} f(n) / g(n) \rightarrow \infty \Longrightarrow f(n) \notin O(g(n))$ y $g(n) \in O(f(n))$

1) $\lim_{n \to \inf.} f(n) / g(n) \in R^+ \implies f(n) \in \Theta(g(n))$

2) $\lim_{n \to \inf.} f(n) / g(n) = 0 \implies f(n) \in O(g(n))$ y $f(n) \notin \Theta(g(n))$

3) $\lim_{n \to \inf.} f(n) / g(n) \to \infty \implies f(n) \notin \Omega(g(n))$ y $f(n) \notin \Theta(g(n))$

Practica 7

Complejidad Temporal

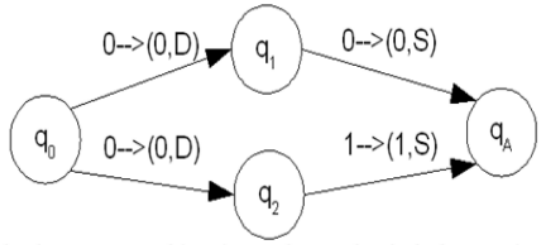
Máquinas de Turing No Determinísticas (MTN)

MT que para un estado y un símbolo barrido por el cabezal, tiene un conjunto finito de alternativas para el siguiente movimiento. Acepta la entrada si en cualquier alternativa se detiene en qA

Las determinísticas son MTD → MTD es un caso particular de una MTN donde el conjunto de alternativas es siempre un conjunto unitario

$M=<Q, \Sigma, \Gamma, \Delta, q_0, q_A, q_R>, \quad \Delta: Q \times \Gamma \to \rho((Q \cup \{q_A, q_R\}) \times \Gamma \times \{D, I, S\})$

Traza de computación. Para una MTN la traza tendrá forma de árbol dónde cada rama representa una computación (una sucesión de alternativas de movimiento)



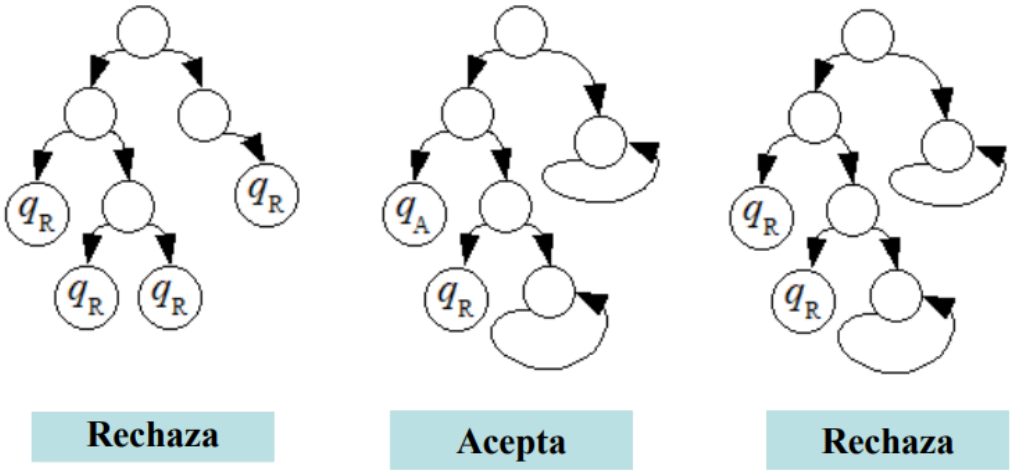
traza para w= 021

$q_0021 \vdash 0q_121 \vdash 0q_R21$

$\vdash 0q_221 \vdash 0q_R21$

M rechaza a 021

La traza es un árbol.



MTN tiene el mismo poder de computo que MTD, se puede hacer una MTD que simule una MTN a traves de un recorrido BFS. Son equivalentes.

Una MTN permite la unión de dos lenguajes.

Sean L1 y L2, dos lenguajes definidos sobre {0,1}*

L1 = $\{0^n 1 \mid n \geq 0\}$

L2 = $\{1^n 0 \mid n \geq 0\}$

a) Construya una MTN M tal que L(M)= L1 ∪ L2
M=<Q, Σ, Γ, Δ, q0, qA, qR>, Δ: Q x Γ → r ((Q U {qA,qR}) x Γ x {D, I, S})

Δ (q0, x) = {(q1, x, S), (q2, x, S)} para todo x ∈ Γ

Δ (q1, 0) = {q1, 0, D)}
Δ (q1, 1) = {q3, 1, D)}

Δ (q2, 1) = {q2, 1, D)}
Δ (q2, 0) = {q3, 0, D)}

Δ (q3, B) = {qA, B, S)}

Todas las otras combinaciones de estado símbolo que faltan llevan al estado qR

Def. Sea M una MTD con k cintas, decimos que M es de complejidad $t(n)$, si para toda entrada de longitud n, M hace a lo sumo $t(n)$ movimientos.

Def. Sea M una MTN con k cintas, decimos que M es de complejidad $t(n)$, si para toda entrada de longitud n, ninguna secuencia de alternativas de movimiento ocasiona que M haga más de $t(n)$ movimientos.

NO SE SUMAN MOVIMIENTOS EN PARALELO.

MTD → Máximo de movimientos

MTN → Ninguna rama hace mas que ese máximo.

Def. Un lenguaje L $\in$ DTIME($t(n)$) sii existe una MTD M, con L=L(M) tal que la complejidad temporal de M pertenece a O($t(n)$)

Def. Un lenguaje L $\in$ NTIME($t(n)$) sii existe una MTN M, con L=L(M) tal que la complejidad temporal de M pertenece a O($t(n)$)

Corolario: DTIME($t(n)$) está incluido en NTIME($t(n)$)

- L ∈ DTIME(t(n)) si existe una MTD M con L=L(M) tal que M pertenece a O(t(n))
- L ∈ NTIME(t(n)) si existe una MTN M con L=L(M) tal que M pertenece a O(t(n))

Def. La clase de lenguajes P comprende todos los lenguajes para los que existe una MTD que lo acepta en tiempo polinomial

$$P = U_{i \geq 0} \, DTIME(n^i)$$

Def. La clase de lenguajes NP comprende todos los lenguajes para los que existe una MTN que lo acepta en tiempo polinomial

$$NP = U_{i \geq 0} \, NTIME(n^i)$$

- Clase P es la union de todos los lenguajes en los que existe una MTD que lo acepta en tiempo polinomial

- Clase NP es la union de todos los lenguajes en los que existe una MTN que lo acepta en tiempo polinomial

P es un subconjunto de NP al ser las MTD un caso particular de MTN

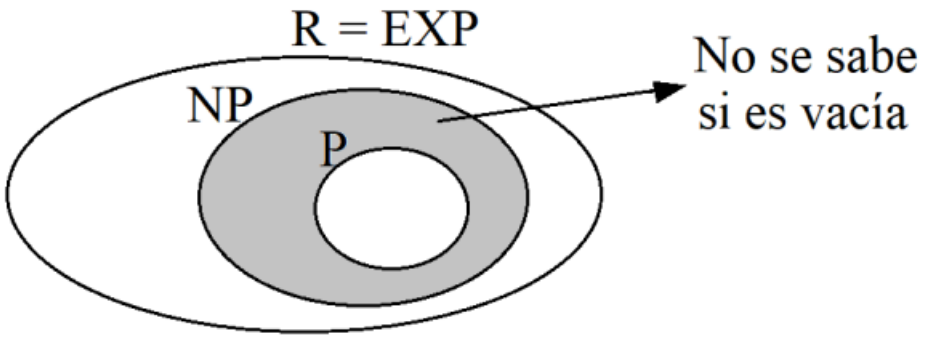
P son los problemas tratables, los que están fuera de P son no tratables → algoritmo no termina o es infinito. No se puede obtener mejor resolución, los no tratables implican muchos pasos de MTD. Las MTD son las máquinas de hoy en día.

Teorema: Si L es un lenguaje recursivo aceptado por una MTN M_1 en tiempo $t(n)$, existe una MTD M_2 que lo acepta en tiempo menor o igual que $d^{t(n)}$, con d una constante mayor que 1 que depende de M_1

Básicamente pertenecer a R no quiere decir que vayas a estar en NP o en P.

R-NP≠∅

No se sabe si P=NP



CO-NP

Definición: CO-NP = $\{ L / \overline{L} \in NP \}$

Teorema 1: Si $L \in P \Rightarrow L \in (NP \cap CO\text{-}NP)$

a) Si $L \in P \Rightarrow L \in NP$ (por def. de P y NP)

b) Si $L \in P \Rightarrow \overline{L} \in P$ (simplemente intercambiando q_A y q_R en la MTD)

$\Rightarrow \overline{L} \in NP$ (por def. de P y NP)

$\Rightarrow L \in CO\text{-}NP$ (por def. CO-NP)

De a) y b) se tiene que $L \in (NP \cap Co\text{-}NP)$

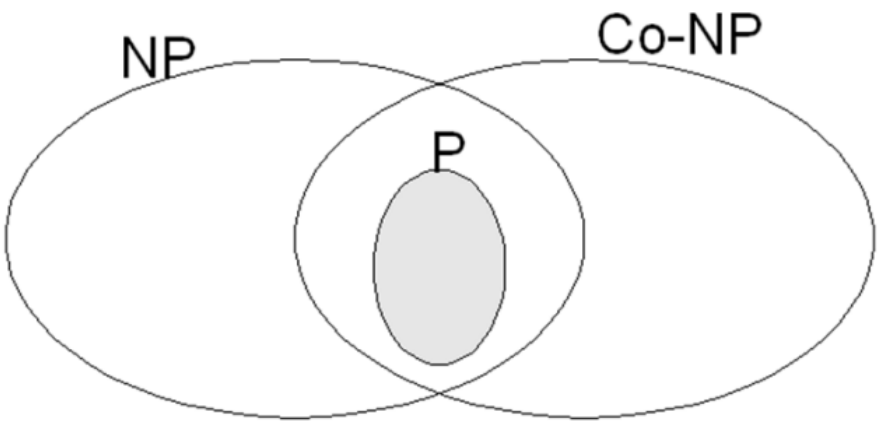
Nota: No se sabe cómo están relacionados NP y CO-NP. No se puede asumir que son iguales. Reconocer el complemento de un lenguaje, intentando intercambiar estados q_A por q_R de su MTN no funciona, una MTN puede aceptar teniendo computaciones que terminan en q_R pues alcanza que una sola termina en q_A.

Teorema 2: $NP \neq Co\text{-}NP \Rightarrow P \neq NP$

Dem: Si $P = NP \Rightarrow NP = CO\text{-}NP$ (porque P = CO-P)

entonces $NP \neq CO\text{-}NP \Rightarrow P \neq NP$ (por contra-recíproca)

Esta es la situación conocida



P=CO-P

NP≠CO-NP

P=NP ∩ CO-NP

Reducción polinomial

L1 αp L2 solo si existe la reducción y la funcion que computa la reduccion (una MTD) trabaja en tiempo polinomial (f pertenece a P)

Teorema 3: $L_1 \alpha_p L_2$ y $L_2 \in P \Rightarrow L_1 \in P$

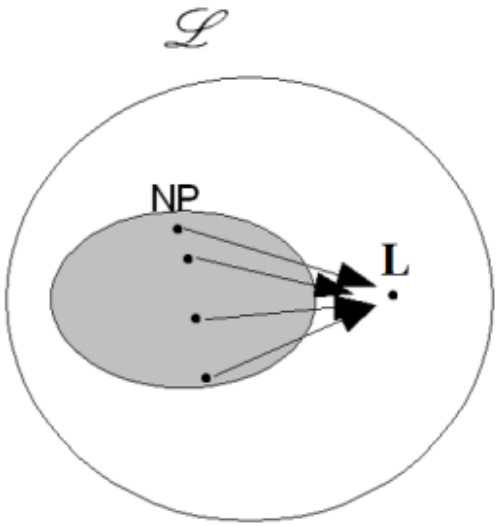
Teorema 4: Sea L un lenguaje tal que $\varnothing \subset L \subset \Sigma^*$ entonces para cualquier lenguaje L' perteneciente a P, vale que $L' \alpha_p L$

Dem: Como $\varnothing \subset L \subset \Sigma^*$ entonces existe algún $x \in L$ y algún $y \notin L$. La reducción consiste para cada w, resolver L' con su MTD polinomial, y asignar x o y según se *acepte* o *rechace* w respectivamente.

No se puede reducir a vacio o a sigma estrella (ACORDATE)

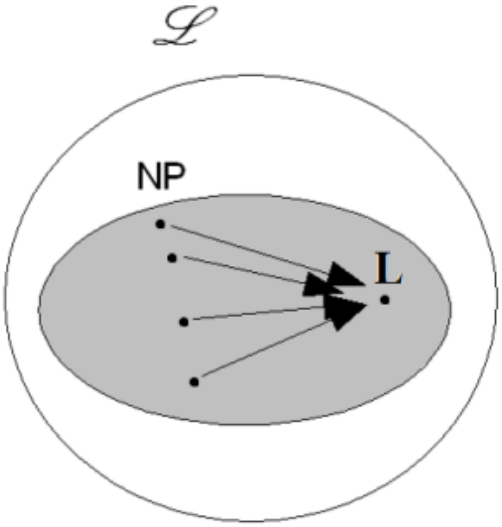
NP-Hard

Definición: $L \in NPH$ (NP-Hard) sii para todo $L' \in NP$ se cumple $L' \alpha_p L$



NP-Completo

Definición: L ∈ NPC (NP-Completo) sii L ∈ NPH y L ∈ NP



NP-Completo

Hasta ahora no se a encontrado ningún lenguaje que cumpla con esto

Teorema 5: Si (L ∈ NPC) AND (L ∈ P) ⇒ P = NP

Dem: Sea L' un lenguaje arbitrario de NP

L' ∈ NP ⇒ L' α_p L (porque por hipótesis L ∈ NPC)

⇒ L' ∈ P (por hiótesis L ∈ P y por teorema 3)

Por lo tanto NP ⊆ P

Por lo tanto P = NP

Nota: Según este teorema para demostrar que P = NP alcanzaría con encontrar una solución polinomial para cualquiera de los problemas NPC conocidos.

NP-Completo

Teorema 6: Sean $L_1, L_2 \in NP$. Si $L_1 \in NPC$ y $L_1 \alpha_p L_2$ entonces $L_2 \in NPC$

Dem: Para todo $L \in NP$ existe $L \alpha_p L_1$ (por $L_1 \in NPC$)
Dado que por hipótesis $L_1 \alpha_p L_2$ se tiene que:
Para todo $L \in NP$ existen $L \alpha_p L_1$ y $L_1 \alpha_p L_2$
Por lo tanto, para todo $L \in NP$ existe $L \alpha_p L_2$ (porque α_p es transitiva)
Por lo tanto $L_2 \in NPH$ (por definición de NPH)
Por lo tanto $L_2 \in NPC$ (por hipótesis $L_2 \in NP$)

αp es:

Reflexivo L1 αp L1 y Transitivo L1 αp L2 ∧ L2 αp L3 ⇒ L1 αp L3.

Practica 8

Análisis

¿Para qué? comparación teórica, escalabilidad, análisis de tiempo y espacio.

Análisis teórico apoyado por el principio de invarianza. Todo lo que se aporte via implementación se acota con una constante.

Análisis de tiempo: el tiempo de ejecución tA(n) de A es la cantidad de pasos, operaciones o acciones elementales que debe realizar el algoritmo al ser ejecutado en una instancia de tamaño n.

Analisis de espacio: el espacio e A(n) de A es la cantidad de datos elementales que el algoritmo necesita al ser ejecutado en una instancia de tamaño n, sin contar la representación de la entrada

Estas son ambiguas por no especificar op elemental ni tamaño.

Operaciones elementales: No dependen de la entrada, acotados por una constante

Diferentes tipos de analisis: mejor, peor, probabilistico

Estructuras de control

En secuencias: Suma del tiempo de ambas secuencias

Tambien aplica la regla del maximo para la notacion asintótica

- Secuencias:

P1; P2 t1(n) y t2(n)

$t1;2(n) = t1(n) + t2(n)$

Regla del máximo

$t1;2(n) = t1(n) + t2(n) \in O(\max(t1(n), t2(n)))$

$t1;2(n) = t1(n) + t2(n) \in \Theta(\max(t1(n), t2(n)))$

Condicional

If (cond)	t1
Then (cuerpo then)	t2
Else (cuerpo else)	t3

Se considera directamente el peor caso:

t1 + max(t2, t3) o directamente

max(t1, t2, t3)

Iteraciones uniformes

For i ← 1 to m

Do P(i) Puede ser P(i, n), t(i)

– Si t(i) no depende de i ==> t(i) = t
  - tfor = m t
  - En cualquier caso, identificar #iter
  - tfor = #iter t

For i ← 1 to m
Do P(i) Puede ser P(i, n), t(i)

– Si t(i) depende de i
  - tfor = $\sum_{i=1}^{m} t(i)$
– Sumas útiles:
  - $\sum_{i=1}^{m} i = \frac{m\,(m+1)}{2}$
  - $\sum_{i=1}^{m} i^2 = \frac{m\,(m+1)\,(2m+1)}{6}$

No hay mejor o peor i

Iteraciones no uniformes

Cantidad de iteraciones desconocida a simple vista

Se puede analizar a través de funciones o recurrencias

Se requiere procesar el step de la función para ver si decrece o no

Son mas complicadas de analizar, puede ser que no se obtenga un resultado preciso dado a la convergencia de las funciones de decrecimiento. El numero de iteraciones puede variar según la condición, el análisis con funciones y/o recurrencias es más complejo. Esta complejidad recae en encontrar que función es la que se reduce. El problema es el análisis de las iteraciones no uniformes y encontrar la función que representa la cantidad de iteraciones. Hay que entrar con el detalle fino de lo que hay que hacer dentro de una iteración no uniforme e incluso hay algunas iteraciones para las cuales no es posible ni si quiera definir la función ni identificar una recurrencia. Esta función, cabe aclarar, que se puede encontrar como una recurrencia o no.

Barometro

Instruccion que se ejecuta más, identificarla y utilizar lo visto en estructura de control una vez se encontro la cantidad de veces que se ejecuta

Solo sirve para saber el Orden o Theta, no sirve para sacar el t(n)

1. Encuentro instrucción que mas se ejecuta.
2. Obtengo la cantidad de veces que se ejecuta.
3. Me permite sacar el orden (es la cantidad de veces que se ejecuta). → O(f(n)) → f(n) es la cantidad de veces que se ejecuta.

Se debe mirar al mas anidado.

Esta no deja de ser detallada (pero no lo es tanto)

Caso Promedio

Se utilizan probabilidades/distribuciones para obtener la t(n) de un problema → complejo

Análisis Amortizado

Es mas relacionado a estructuras de control donde es poco probable que en todas las llamadas se tenga el peor caso.

Muy especifico a ciertos casos/algoritmos

Alternativa a mejor/peor/probabilístico

Básicamente la cantidad de veces de operación depende del estado de la estructura.

Recurrencias

Funciones dadas en funcion de si mismas achicando cada vez mas la entrada hasta llegar a un caso base

Diferentes estrategias:

- D&C (Divide and conquer)

- D&S/C-C (Divide and solve/Conquer-combine)

De recurrencia no homogénea puedo llegar a homogénea.

La recurrencia en general se demuestra por inducción.

DOS RECETAS

Dos “recetas”

– $t(n) = a\, t(n-b) + f(n)$ (reduce restando)

– $t(n) = a\, t(n/b) + f(n)$ (reduce dividiendo)

- a: cantidad de llamadas recursivas
- b: constante
- f(n): operaciones “extra” llamadas recursivas

Recursión, n ==> n-b

– $t(n) = \begin{cases} c & n \leq b \\ a\,t(n-b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$

– $t(n) \in \begin{cases} O(n^{k+1}) & \text{si } a = 1 \\ O(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$

– Vale con todos Θ( )

Recursión, n ==> n/b

– $t(n) = \begin{cases} c & n \leq b \\ a\,t(n/b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$

– $t(n) \in \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(n^k) & \text{si } a < b^k \end{cases}$

– Vale con todos Θ( ). Teorema Maestro o Método Maestro

No todas las recurrencias estan cubiertas por estas recetas

1) Considerar las matrices A, B, C ∈ $IR^{(n \times n)}$, 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

$$
\begin{aligned}
C_{1,1} &= A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1} \\
C_{1,2} &= A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2} \\
C_{2,1} &= A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1} \\
C_{2,2} &= A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2}
\end{aligned}
$$

¿Sería necesario definir algo más?

Sería necesario definir las llamadas recursivas de los casos en donde la matriz no es 2x2 y aclarar que se realiza cuando si son 2x2.

Utilizando la receta calculamos T(n)

T(0) = 1
T(n) = 8T(n/2) + f(n^2).

a = 8 b=2 f(n)=n^k k=2

a > b^k → a > 2^2 → a > 4

$\Theta(n^{\log b(a)})$ → $\Theta(n^{\log 2(8)})$ → $\Theta(n^3)$

b) Buscar el algoritmo de Strassen, dar su definición en función de las submatrices Xi,j anteriores y dar su orden de tiempo de ejecución.

Video a partir del minuto 5:50 honestamente lo explica mejor de lo que yo jamás podría.

c) Comparar los dos algoritmos de multiplicación de matrices. ¿Los dos son algoritmos D&C? ¿Alguno de los dos es “mejor” que el otro en cuanto a tiempo de ejecución?

Ambos utilizan D&C, pero el de Strassen es “mejor” ya que su tiempo de ejecución es de $\Theta(n^{2.81})$ → $\Theta(n^{\log 2(7)})$

Diseño + Analisis

Mas asociados al diseño que al analisis.

– Divide and Conquer

recursivos ==> recurrencias

– Algoritmos Greedy

Problemas de optimizacion

Construyen la solucion paso a paso

Toman la mejor opcion del momento → Puede que a la larga no sea la mejor. No es necesariamente la mejor solución, si no que da una rápida en el momento.

– Programación dinámica

Estrategia bottomup

Se resuelve problemas mas sencillos primero, luego lo combino y resuelvo lo mas grande.

Se reutilizan resultados intermedios (tablas o hashes)

– Algoritmos probabilísticos

Cuando se debe tomar una decisión, se toma al azar, no se computan costos p/ evaluar.
A mayor generacion de aleatorios menor error

Ejemplo → obtener área de superficie irregular.

El análisis es muy específico de cada algoritmo más que del propio diseño.