

Tareas Complejidad y Computabilidad

Dalia Camacho

Lenguajes

1) $A(BC) \subseteq (AB)C$

Sea $x \in A(BC) \Rightarrow x = x_1 x_2$, donde $x_1 \in A$ y $x_2 \in BC$. Como $x_2 \in BC$, $x_2 = y_1 y_2$, donde $y_1 \in B$ y $y_2 \in C$. $\Rightarrow x = x_1 y_1 y_2 = (x_1 y_1) y_2$. Como $x_1 \in A$ y $y_1 \in B$ $x_1 y_1 = z$, $z \in AB$. $\Rightarrow x = z y_2$. Como $z \in AB$ y $y_2 \in C$ $x \in (AB)C$. $\therefore A(BC) \subseteq (AB)C$ \blacksquare

2) $A\{\lambda\} = \{\lambda\}A = A$

Demostraremos $A\{\lambda\} \subseteq \{\lambda\}A$

Sea $x \in A\{\lambda\} \Rightarrow x = x_1 \lambda$, donde $x_1 \in A$ y λ la cadena vacía.

Como λ es la cadena vacía $x = x_1 \lambda = x_1$, además $\lambda x_1 = x_1$

$\therefore x = x_1 \lambda = x_1 = \lambda x_1$, como $x_1 \in A$, $\lambda x_1 \in \{\lambda\}A$ $\therefore x \in \{\lambda\}A$
 $\therefore A\{\lambda\} \subseteq \{\lambda\}A$.

Ahora mostramos $\{\lambda\}A \subseteq A$.

Sea $x \in \{\lambda\}A \Rightarrow x = \lambda x_1$, donde λ es la cadena vacía y $x_1 \in A$.

Como λ es la cadena vacía $\lambda x_1 = x_1 \therefore x = x_1$ y $x \in A$

$\therefore \{\lambda\}A \subseteq A$.

Ahora probamos $A \subseteq A\{\lambda\}$

Sea $x \in A$ y λ la cadena vacía $\Rightarrow x = x \lambda \therefore x \in A\{\lambda\}$

$\therefore A \subseteq A\{\lambda\}$

Con esto tenemos que

$A\{\lambda\} \subseteq \{\lambda\}A \subseteq A \subseteq A\{\lambda\}$, como $A\{\lambda\} = \{\lambda\}A$, se tiene
que $A\{\lambda\} = \{\lambda\}A = A$ \blacksquare

Cerraduras de Kleene

1) P.D. $A^* A^* = A^*$

Sea $x \in A^* A^* \Rightarrow x = yz$, donde $y \in A^*$ y $z \in A^*$
 $\Rightarrow y = y_1 y_2 y_3 \dots y_m$, donde $y_i \in A$ y $z = z_1 z_2 \dots z_n$ donde $z_i \in A$
 $\therefore x = y_1 y_2 y_3 \dots y_m z_1 z_2 z_3 \dots z_n$, tal que cada símbolo en
 x es elemento de $A^* \therefore x \in A^* \therefore A^* A^* \subseteq A^*$

Sea $x \in A^*$ y sea λ la cadena vacía, sabemos que

$x = x\lambda$ y que $\lambda \in A^* \therefore x\lambda \in A^* A^* \therefore x \in A^* A^* \therefore A^* \subseteq A^* A^*$

Con esto se tiene que $A^* A^* \subseteq A^* \subseteq A^* A^* \therefore A^* A^* = A^* \square$

2) P.D. $A^{**} = A^*$

Sea $x \in A^{**} \Rightarrow x = x_1 x_2 \dots x_m$, donde $x_i \in A^*$, por lo que cada x_i es de la forma $x_i = x_{i1} x_{i2} \dots x_{ik_i}$, donde $x_{ij} \in A$. Entonces
 $x = x_1 x_2 \dots x_m$, donde $x_i = x_{i1} x_{i2} \dots x_{ik_i}$ y cada símbolo
de x es elemento de $A \therefore x \in A^* \therefore A^{**} \subseteq A^*$.

Sea $x \in A^*$, sabemos quedado cualquier alfabeto Σ y cualquier
 $y \in \Sigma$, $y \in \Sigma^* \therefore$ si $x \in A^*$ entonces $x \in (A^*)^* = A^{**} \therefore A^* \subseteq A^{**}$

Con esto tenemos que $A^{**} \subseteq A^* \subseteq A^{**} \therefore A^* = A^{**}$

3) P.D. $A^* = \{\lambda\} \cup A^* = \{\lambda\} \cup A^* A$

Sea $x \in A^* \Rightarrow x \in A^*$ o $x \in \{\lambda\} \Rightarrow x \in \{\lambda\} \cup A^*$

$\therefore A^* \subseteq \{\lambda\} \cup A^*$

Sea $x \in \{\lambda\} \cup A^* \Rightarrow x = \lambda$ o $x \in A^*$

(Caso 1) Si $x = \lambda \Rightarrow x \in \{\lambda\} \Rightarrow x \in \{\lambda\} \cup A^* A$.

Caso 2: $x \in A^* \Rightarrow x = x_1 x_2 \dots x_m = (x_1 x_2 \dots x_{m-1}) x_m$, donde

$(x_1 x_2 \dots x_{m-1}) \in A^*$ y $x_m \in A \therefore x \in A^* A \therefore x \in \{\lambda\} \cup A^* A$.

O bien $x = x_1$, $x_1 \in A$, sea λ la cadena vacía $\Rightarrow \lambda x_1 = x_1$.

Como $\lambda \in \Sigma^*$ para cualquier alfabeto Σ , $\lambda \in A^* \therefore x = \lambda x_1 \in A^* A$
 $\therefore x \in \{\lambda\} \cup A^* A$.

Como ya cubrimos todos los casos se tiene que

$$\{\lambda\} \cup A^* \subseteq \{\lambda\} \cup A^* A.$$

Sea $x \in \{\lambda\} \cup A^* A \Rightarrow x = \lambda \circ x \in A^* A$.

Caso $x = \lambda$: Si $x = \lambda \in A^*$ ya que para cualquier alfabeto Σ , $\lambda \in \Sigma^*$.

Caso $x \in A^* A$: Si $x \in A^* A \Rightarrow x = x_1 x_2 \dots x_m y$, donde

$(x_1 x_2 \dots x_m) \in A^*$ y $y \in A$. Como $(x_1 x_2 \dots x_m) \in A^*$ $x_i \in A$

$\forall i \in \{1, 2, \dots, m\}$. $\therefore x = x_1 x_2 \dots x_m y$ tiene todos sus símbolos en $A \therefore x \in A^*$

Como ya se cubrieron los casos posibles $\{\lambda\} \cup A^* A \subseteq A^*$

Con esto tenemos que $A^* \subseteq \{\lambda\} \cup A^* \subseteq \{\lambda\} \cup A^* A \subseteq A^*$

$$\therefore A^* = \{\lambda\} \cup A^* = \{\lambda\} \cup A^* A \quad \square$$

Máquinas de Turing

Dado $\Sigma = \{a, b\}$, define una MT que acepte el lenguaje

$$L = \{ww \mid w \in \Sigma^*\}$$

$$\text{Definimos } T = \{a, b, \leftarrow, \rightarrow, \downarrow, a^*, b^*\}$$

La función de transición se muestra en la siguiente página.

Delta	-	a	b	-	-	a*	b*
q_inicial	q_lambda, -, R	-	-	-	-	-	-
q_lambda	-	q_parA, a, L	q_parA, b, L	-	qs, -, -	-	-
q_parA	q_parA, -, R	q_imparA, a, R	q_parA, b, R	-	q_parB, -, L	-	-
q_imparA	-	q_parA, a, R	q_imparA, b, R	-	qn, -, -	-	-
q_parB	q_buscarS, -, R	q_parB, a, L	q_imparB, b, L	-	-	-	-
q_imparB	qn, -, -	q_imparB, a, L	q_parB, b, L	-	-	-	-
q_buscarS	-	q_final_A, _, R	q_finalB, _, R	-	q_buscarS, _, R	qs, -, -	qs, -, -
q_final_A	-	q_final_A, a, R	q_final_A, b, R	q_comparaA, -, L	-	-	-
q_final_B	-	q_final_B, a, R	q_final_B, b, R	q_comparaB, -, L	-	-	-
q_comparaA	-	q_final, a*, R	q_escribeA, b, R	-	qn, -, -	q_comparaA, a*, L	q_comparaA, b*, L
q_comparaB	-	q_escribeB, a, R	q_final, b*, R	-	qn, -, -	q_comparaB, a*, L	q_comparaB, b*, L
q_final	-	q_busca_ _a, _, L	q_busca_ _b, _, L	q_final, -, R	q_rectifica, _, L	q_final, a*, R	q_final, b*, R
q_escribeA	-	q_escribeA, a, R	q_escribeA, b, R	q_escribeA, -, R	q_reescribe, a, L	q_escribeA, a*, R	q_escribeA, b*, R
q_escribeB	-	q_escribeB, a, R	q_escribeB, b, R	q_escribeB, -, R	q_reescribe, b, L	q_escribeB, a*, R	q_escribeB, b*, R
q_busca_ _a	-	q_busca_ _a, a, L	q_busca_ _a, b, L	q_comparaA, -, L	-	-	-
q_busca_ _b	-	q_busca_ _b, a, L	q_busca_ _b, b, L	q_comparaB, -, L	-	-	-
q_rectifica	q_buscarS, -, R	q_reescribe, a, R	q_reescribe, b, R	q_rectifica, -, L	qs, -, -	q_rectifica, a*, L	q_rectifica, b*, L
q_reescribe	-	q_reescribe, a, R	q_reescribe, b, R	q_regresas, -, L	-	q_reescribeA, a, R	q_reescribeB, b, R
q_reescribeA	-	q_reescribeA, a, R	q_reescribeA, b, R	q_reescribeA, -, R	q_reescribe_sig, a, L	q_reescribeA, a*, R	q_reescribeA, b*, R
q_reescribeB	-	q_reescribeB, a, R	q_reescribeB, b, R	q_reescribeB, -, R	q_reescribe_sig, b, L	q_reescribeB, a*, R	q_reescribeB, b*, R
q_reescribe_sig	-	q_reescribe_sig, a, L	q_reescribe_sig, b, L	q_reescribe_sig, -, L	q_reescribe, _, R	q_reescribe_sig, a*, L	q_reescribe_sig, b*, L
q_regresas	-	q_regresas, a, L	q_regresas, b, L	q_regresas, -, L	q_buscarS, _, R	-	-

Lo primero que se hace es verificar si se trata de la cadena vacía, si sí, se pasa al estado de aceptación qs. Si no se regresa al principio y verifica que haya un número par de a's, si no se pasa al estado de rechazo qn. Si sí se escribe + al final de la cadena. Después se checa que haya un número par de b's, yendo hacia la derecha. Si hay un número impar de b's se va al estado qn. Si no se lee el primer símbolo no vacío, se elimina y se compara con el último símbolo de la cadena. Si los símbolos no coinciden, el símbolo leído inicialmente se escribe en el primer espacio vacío después de +; y se busca el primer símbolo no vacío después de + para repetir el proceso. Si sí coinciden, entonces se reescribe en su versión * (si es a se escribe a* y si es b se escribe b*). Y buscamos el último símbolo no vacío después de +. Si ya no hay símbolos después de +, verificamos que entre + y + sólo haya espacios o símbolos en la forma *. Si sí entonces se entra al estado qs. Si no se reescriben todos los símbolos de la forma * →

a su forma original en la cadena y después de \vdash . Una vez que se han reescrito, se repite el proceso desde la búsqueda del primer símbolo distinto del vacío después de \vdash .

En cambio si S hay otros símbolos por checar después de \vdash entonces se comparan con el primer símbolo (de izq a derecha) después de \vdash ; y se repite el proceso.

2) Demostrar que el conjunto de máquinas de Turing es numerable.

Dada una MT podemos encontrar una codificación finita sobre $\Sigma = \{0, 1\}$. Por ejemplo si enumeramos los estados en q y al estado i le corresponde la cadena O^i y al símbolo j le corresponde la cadena O^j , podemos definir a la MT con la siguiente codificación.

$10^1 10^2 1 \dots 0^{m-1} 10^m 110^1 10^2 \dots 0^{n-1} 10^n 110^1 110^j 110^k$

$\underbrace{\hspace{1cm}}_{\text{inicio}}$ $\underbrace{\hspace{1cm}}_Q$ $\underbrace{\hspace{1cm}}_T$ $\underbrace{\hspace{1cm}}_{q_0}$ $\underbrace{\hspace{1cm}}_{q_s}$ $\underbrace{\hspace{1cm}}_{q_n}$

$110^1 10^1 10^l 10^r 10^s 110^2 10^1 10^l 10^r 10^s 10^t 11 \dots$

$\delta(q_1, s_1) = (q_l, s_p, R)$

$\dots 110^m 10^n 10^l 10^r 10^s 10^t 10^u 10^v 10^w 10^x 10^y 10^z \dots$

En este caso el primer 1 define que inicia la MT y se definen los estados separados por un 1. Cuando hay 2 unos, se inicia la definición de los símbolos en T , separadas por un 1.

Cuando hay dos unos se indica que definimos q_0, q_s, q_n , después con 2 unos se marcan las evaluaciones de δ , donde $l=0$ y $R=00$. Como T y Q son finitas δ es finita y la codificación es una cadena finita.

Ahora definimos $f: \Sigma^* \rightarrow \mathbb{N}$, tal que f traduce cadenas binarias a su representación en base 10. Sabemos que f es biyectiva y que la codificación definida anteriormente es \subseteq de Σ^* . $\therefore |\text{MT}| \leq |\Sigma^*| = |\mathbb{N}|$.

\therefore El conjunto de MT es numerable. \square

3) Demostrar que el conjunto de todos los lenguajes es no numerable.

Supongamos que el conjunto de todos los lenguajes es numerable, entonces el conjunto de todos los lenguajes sobre un alfabeto Σ es numerable. Sean s_1, s_2, \dots la enumeración de todas las posibles cadenas de Σ^* .

Entonces el conjunto de todos los lenguajes sobre Σ se puede representar con la siguiente enumeración

	s_1	s_2	s_3	s_4	\dots
L_1	1	1	0	0	\dots
L_2	0	1	0	0	\dots
L_3	0	1	0	1	\dots
\vdots					

Donde cada renglón corresponde a un lenguaje y el 1 o 0 indican si la cadena s_i pertenece al lenguaje j .

Dada esa enumeración consideremos el lenguaje nL sobre Σ que toma cada elemento de la diagonal de forma opuesta. Es decir si: $s_i \in L_i \Rightarrow s_i \notin nL$ y si: $s_i \notin L_i \Rightarrow s_i \in nL \quad \forall i \in \mathbb{N}$. Por construcción $nL \neq L_i \quad \forall i \in \mathbb{N} \therefore nL$ no está en la enumeración y el conjunto de los lenguajes sobre Σ no es numerable! \square

\therefore El conjunto de lenguajes es no numerable. \square

4) Concluir que existen más problemas que soluciones.

Tenemos que el conjunto de MT es numerable y que el conjunto de lenguajes es no numerable $\therefore |MT| < |\text{Lenguajes}|$
 \therefore Hay más lenguajes que MT \therefore Hay más problemas de los que se pueden resolver. \square

5) Definir una máquina de Turing que sume $x, y \in \mathbb{N} \cup \{\lambda\}$.

Sea $\Sigma = \{0, 1, \lambda\}$, tal que $1^n = n$ y $\lambda = 0$, además sea

$f: \Sigma^* \rightarrow \Sigma^*$ tal que $f(1^n 0 1^m) = 1^{n+m}$; $f(1^n 0 \lambda) = 1^n$;

$f(\lambda 0 1^m) = 1^m$; y $f(\lambda 0 \lambda) = \lambda$. Dado $T = \{ \leftarrow, 0, 1, \lambda, - \}$

Se puede definir la siguiente función f tal que en caso de aceptación el resultado es la suma de 2 naturales.

delta	-	1	0	lambda	-
q0	q1, -, R	-	-	-	-
q1	-	1suma_q1, 1, R	qn, -, -	0suma_q1, 1, R	qn, -, -
1suma_q1	-	1suma_q1, 1, R	1suma_q2, 1, R	qn, -, -	qn, -, -
1suma_q2	-	1suma1_q1, 1, R	qn, -, -	fin_no_0	qn, -, -
1suma1_q1	-	1suma1_q1, 1, R	qn, -, -	qn, -, -	fin_no_0
fin_no_0	-	qs, _, -	-	-	-
0suma_q1	-	0suma1_q1, 1, R	qn, -, -	0suma0_q1, _, L	qn, -, -
0suma1_q1	-	0suma1_q1, 1, R	qn, -, -	qn, -, -	0suma1_q2, _, L
0suma1_q2	-	fin_no_0	-	-	-
0suma0_q1	-	0suma0_q2, _, L	-	-	-
0suma0_q2	-	qs, lambda, -	-	-	-
qn	-	-	-	-	-
qs	-	-	-	-	-

Lenguajes computables y computables enumerablemente.

1) P.D. $HP \in C.E.$

$$HP = \{ M \# x \mid M \text{ se detiene en } x \}$$

Sea N una MT que acepta $M \# x$ si M se detiene en x . Supongamos que $HP \notin C.E.$ $\Rightarrow \exists M \# x \in HP$ que no es reconocido por $N \Rightarrow$

M no se detiene en $x \therefore M \# x \notin HP \quad \nabla$

$\therefore HP \in C.E. \quad \square$

2) P.D. $MP \in C.E.$

$$MP = \{ M \# x \mid x \in L(M) \}$$

Sea N una MT que acepta $M \# x$ si M acepta a x . Supongamos que $MP \notin C.E.$ $\Rightarrow \exists M \# x \in MP$ tal que $M \# x$ no es reconocido por $N \Rightarrow M$ no acepta a $x \Rightarrow x \notin L(M) \Rightarrow M \# x \notin MP \quad \nabla$

$\therefore MP \in C.E. \quad \square$

3) Probar que $\nexists A$ y $P(A)$ no existe $f: A \rightarrow P(A)$ t.c. f sea biyectiva.

Para que f sea biyectiva, f debe ser suprayectiva y haciendo uso del Teorema de Cantor se tiene que $\nexists f: A \rightarrow P(A)$ t.c. A sea suprayectiva. Esto se demuestra de la siguiente manera:

Supongamos $\exists f: A \rightarrow P(A)$ suprayectiva y sea $B \subseteq A$ t.c.

$B = \{ x \in A \mid x \notin f(x) \}$. Como f es suprayectiva $\Rightarrow \exists a \in A$ t.c.

$f(a) = B$. Si $a \in B \Rightarrow a \notin f(a) \Rightarrow a \in B \quad \nabla$

Si $a \notin B \Rightarrow a \notin f(a) \Rightarrow a \in B \quad \nabla$

$\therefore \nexists a \in A$ t.c. $f(a) = B$. $\therefore \exists B \in P(A)$ que no está en la imagen

de f . $\therefore f$ no es suprayectiva.

Como no existe $f: A \rightarrow P(A)$ suprayectiva $\Rightarrow \nexists f: A \rightarrow P(A)$ biyectiva. \square

4) Probar que el siguiente conjunto es computable

$$HP_T = \{ M \# x \# t \in \{0,1\}^* \mid M \text{ se detiene en } x \text{ en a lo más } t \text{ pasos} \}$$

Sea N una MT que acepta si M se detiene en x en a lo más t pasos y rechaza si M no se ha detenido en $t+1$ pasos. ∴

S: $M \# x \# t$ se detiene en a lo más t pasos N acepta.

S: $M \# x \# t$ no se detiene en x en t paso, N rechaza.

Si la cadena de entrada no es de la forma $M \# x \# t$, entonces se rechaza, no se encicla. porque esto puede verificarse en tiempo finito.

∴ N acepta una cadena l si: $l \in HP_T$ y rechaza en otro caso.

∴ N no se encicla y $HP_T = L(N)$

∴ HP_T es computable \blacksquare

Conjuntos co-CE, relaciones decidibles y funciones computables

1. Probar que $A \in \text{co-CE} \Leftrightarrow \exists R(\cdot, \cdot)$ relación binaria decidable

$$\therefore A = \{x \in \Sigma^* \mid \nexists y \in \Sigma^* R(x, y)\}$$

⇒) Sabemos que A es co-CE $\Rightarrow A^c$ es C.E. y $\exists R_M(\cdot, \cdot)$

decidable. ∵ $A^c = \{x \in \Sigma^* \mid \exists y, R_M(x, y)\}$. Podemos definir $R(\cdot, \cdot)$ como la relación binaria tal que:

$$R(x, y) = \begin{cases} \text{verdadero si } R_M(x, y) = \text{falso} \\ \text{falso si } R_M(x, y) = \text{verdadero} \end{cases}$$

como $R_M(x, y)$ es decidable, $R(x, y)$ también lo es.

Además si: $x \notin A^c \Rightarrow \nexists y \in \Sigma^* \text{ s.t. } R_M(x, y) \text{ sea verdadero} \Rightarrow$

$\forall y \in \Sigma^* R_M(x, y)$ es falso ∴ $\forall y \in \Sigma^* R(x, y)$ es falso para $x \notin A^c$ ($x \in A$).

∴ S: $A \in \text{co-CE} \Rightarrow \exists R(\cdot, \cdot)$ decidable ∵ $A = \{x \in \Sigma^* \mid \forall y \in \Sigma^*, R(x, y)\}$

\Leftarrow) Sea $R(\cdot, \cdot)$ decidable t: $A = \{x \in \Sigma^* \mid \exists y R(x, y)\}$
 consideremos el conjunto G definido por $G = \{x \in \Sigma^* \mid \exists y R_2(x, y)\}$

Donde

$$R_2 = \begin{cases} \text{verdadero, } R(x, y) \text{ es falso} \\ \text{falso, } R(x, y) \text{ es verdadero} \end{cases}$$

Además sabemos que G es C.E. Ahora basta probar que $G = A^c$.

Si $x \in G \Rightarrow \exists y \nexists R_2(x, y)$ es verdadera $\therefore \exists y \nexists R(x, y)$ es falso $\therefore x \notin A \Rightarrow x \in A^c \therefore G \subseteq A^c$.

Sea $x \in A^c \Rightarrow \exists y \nexists R(x, y)$ es falso $\Rightarrow \exists y \nexists R_2(x, y)$ es verdadero $\therefore x \in G \therefore A^c \subseteq G \therefore A^c = G$

Con esto tenemos que A^c es C.E $\therefore A$ es co-C.E



2. Probar que $A \subseteq \Sigma^*$, A es C.E $\Leftrightarrow A$ es dom f para alguna función computable $f: \subseteq \Sigma^* \rightarrow \Sigma^*$

\Rightarrow) Sea A C.E $\Rightarrow \exists R(x, y)$ decidable t:

$A = \{x \in \Sigma^* \mid \exists y \in \Sigma^* \exists R(x, y)\}$. Con esto se puede definir $f: A \subseteq \Sigma^* \rightarrow \Sigma^*$ como $f(x) = y$, donde y es el primer testigo de $R(x, \cdot)$ en orden lexicográfico. Además, $R(\cdot, \cdot)$ es decidable y para cualquier $x \in A$ es posible encontrar a y el primer testigo de x bajo $R(\cdot, \cdot)$ $\Rightarrow f$ es computable para cualquier $x \in A$. $\therefore \exists f: \subseteq \Sigma^* \rightarrow \Sigma^*$ computable cuyo dominio es A .

\Leftarrow) Sea $f: \subseteq \Sigma^* \rightarrow \Sigma^*$ y A dominio de f . Sea

$$R(x, y) = \begin{cases} V, & s: f(x) = y \\ F, & e.o.c. \end{cases}$$

S: $x \in A \Rightarrow \exists y = f(x) \nexists R(x, y)$ es verdadero \Rightarrow
 $x \in \{x \in \Sigma^* \mid \exists y \in \Sigma^* R(x, y)\} \therefore A \subseteq \{x \in \Sigma^* \mid \exists y R(x, y)\}$.

Si $x \in \{x \in \Sigma^* \mid \exists y R(x, y)\} \Rightarrow \exists y \text{ s.t. } R(x, y) \text{ es verdadero}$
 por definición de $R(x, y)$, $f(x)=y$ y puede calcularse \therefore
 $x \in A \Rightarrow A = \{x \in \Sigma^* \mid \exists y R(x, y)\}$ para alguna $R(\cdot, \cdot)$ decidable
 $\Rightarrow A \in C.E.$



Reducciones

1. Probar que las relaciones de reducción son transitivas.

Sean A, B y C lenguajes tales que $A \leq_m B$ y $B \leq_m C$,
 entonces $\exists \delta_1: \Sigma^* \rightarrow \Sigma^*$ computable s.t. $\delta_1(x) \in B \Leftrightarrow x \in A$ y
 $\delta_2: \Sigma^* \rightarrow \Sigma^*$ computable s.t. $\delta_2(x) \in C \Leftrightarrow x \in B$. Con esto podemos
 construir $\delta: \Sigma^* \rightarrow \Sigma^*$, $\delta(x) = \delta_2(\delta_1(x)) \Rightarrow x \in A \Leftrightarrow \delta_1(x) \in B$,
 $\text{y } \delta_1(x) \in B \Leftrightarrow \delta_2(\delta_1(x)) \in C \Rightarrow x \in A \Leftrightarrow \delta(x) \in C$
 $\therefore \exists \delta \text{ s.t. } x \in A \Leftrightarrow \delta(x) \in C \quad \blacksquare$

2. Probar que las relaciones de reducción son reflexivas

Sea A un lenguaje $\Rightarrow \exists \delta: \Sigma^* \rightarrow \Sigma^*$ s.t. $\delta(x) = x$, si
 $x \in A \Rightarrow \delta(x) = x \in A$ y si $y \notin A \Rightarrow \delta(y) = y \notin A \Rightarrow \exists \delta$
 computable s.t. $\delta(x) \in A \Leftrightarrow x \in A \therefore A \leq_m A \quad \blacksquare$

Complejidad

Notación asintótica

1. Probar que dadas f y g funciones,

$$f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in \Omega(g(n)) \text{ y } f(n) \in O(g(n)).$$

\Rightarrow Sea $f(n) \in \Theta(g(n)) \Rightarrow \exists c_1, c_2 > 0$ y $n_0 \in \mathbb{N}$ s.t.

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0.$$

$\Rightarrow \exists c = c_1$ y n_0 s.t. $0 \leq c g(n) \leq f(n) \quad \forall n \geq n_0 \therefore f(n) \in \Omega(g(n))$.

y $\exists c = c_2$ y n_0 s.t. $f(n) \leq c g(n) \quad \forall n \geq n_0 \therefore f(n) \in O(g(n))$.

\Leftarrow Sea $f \neq f \in \Omega(g(n))$ y $f \in O(g(n)) \Rightarrow \exists c_1 > 0$ y $n_1 \in \mathbb{N}$

$\exists 0 \leq c_1 g(n) \leq f(n) \quad \forall n \geq n_1$ y $\exists c_2 > 0$ y $n_2 \in \mathbb{N} \quad \exists f(n) \leq c_2 g(n)$

$\forall n \geq n_2 \therefore \exists n_0 = \max\{n_1, n_2\} \quad \exists 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n),$

$\forall n \geq n_0 \therefore f \in \Theta(g(n)) \quad \square$

Más notación asintótica

1. Probar los siguientes

a) $n^3 \neq \Theta(n^4)$

Supongamos $n^3 = \Theta(n^4) \Rightarrow n^3 = \Omega(n^4) \Rightarrow \exists c > 0$ y $n_0 \in \mathbb{N} \quad \exists$

$c n^4 \leq n^3 \quad \forall n \geq n_0$ (SPG $n_0 > 0$) $\Rightarrow c n \leq 1 \Rightarrow n \leq \frac{1}{c} \quad \forall n > n_0$.

Sea $n = \lceil \frac{1}{c} \rceil + 1 \Rightarrow n > \frac{1}{c}$ y $n \leq \frac{1}{c}$ \square

$\therefore n^3 \neq \Theta(n^4) \quad \square$

b) $\log_2 n = \Theta(n)$

Hay que encontrar $c > 0$ y $n_0 \in \mathbb{N} \quad \exists \log_2 n \leq cn \quad \forall n \geq n_0$

$\log_2 n \leq cn$

Sea $n_0 = 2$ y $c = 1 \Rightarrow \log_2 2 = 1 \leq 2 \Rightarrow \log_2(2) \leq c \cdot 2$, además

n crece más rápido que $\log_2 n \Rightarrow$ para $c = 1$ y $n_0 = 2$

$\log_2 n \leq cn \quad \forall n \geq n_0 \quad \square$

c) $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$

Buscamos c y n_0 $\exists \max\{f(n), g(n)\} \leq c(f(n) + g(n)) \quad \forall n \geq n_0$

Si $f, g : \mathbb{N} \rightarrow \mathbb{N} \Rightarrow f(n) \leq f(n) + g(n) \quad \forall n \in \mathbb{N}$ y

$g(n) \leq f(n) + g(n) \quad \forall n \in \mathbb{N} \Rightarrow \max\{f(n), g(n)\} \leq f(n) + g(n)$

$\forall n \in \mathbb{N}$. Entonces para $c = 1$, $n_0 = 1$ se tiene que

$\max\{f(n), g(n)\} \leq c(f(n) + g(n)) \quad \forall n \geq n_0 \therefore$

$\max\{f(n), g(n)\} \in \Theta(f(n) + g(n)) \quad \square$

2. Probar los siguientes

a) $f(n) = O(g(n)) \text{ y } g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$

Sea $f(n) = O(g(n)) \text{ y } g(n) = O(h(n)) \Rightarrow \exists c_1 \text{ y } n_1 \text{ s.t.}$

$$f(n) \leq c_1 g(n) \quad \forall n \geq n_1 \text{ y } \exists c_2 \text{ y } n_2 \text{ s.t. } g(n) \leq c_2 h(n)$$

$$\forall n \geq n_2 \Rightarrow f(n) \leq c_1 g(n) \leq c_1 \cdot c_2 h(n) \quad \forall n \geq \max\{n_1, n_2\}$$

$$\Rightarrow \exists n_0 = \max\{n_1, n_2\} \text{ y } c = c_1 \cdot c_2 \text{ s.t. } f(n) \leq c h(n) \quad \forall n \geq n_0$$

$$\therefore f(n) \in O(h(n)) \quad \square$$

b) $f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$

Sea $f(n) = O(g(n)) \Rightarrow \exists c > 0 \text{ y } n_0 \in \mathbb{N} \text{ s.t.}$

$$f(n) \leq c g(n) \quad \forall n \geq n_0, \text{ Como } c > 0 \Rightarrow \frac{1}{c} f(n) \leq g(n)$$

$$\forall n > n_0 \Rightarrow \exists c' = \frac{1}{c} > 0 \text{ y } n'_0 = n_0 \cdot c' \quad c' f(n) \leq g(n)$$

$$\forall n > n'_0 \therefore g(n) \in \Omega(f(n)) \quad \square$$

3. Decir Verdadero o Falso y por qué

a) $2^{n+1} = O(2^n)$? Verdadero

$$2^{n+1} = 2 \cdot 2^n \Rightarrow \text{para } c=2 \text{ y } n_0=1 \text{ se tiene que}$$

$$2^{n+1} \leq c \cdot 2^n \quad \forall n \geq n_0$$

b) $2^{2^n} = O(2^n)$? Falso

Suponemos que $\exists c > 0 \text{ y } n_0 \in \mathbb{N} \text{ s.t. } 2^{2^n} \leq c \cdot 2^n$

$$\Rightarrow 2^n \leq \log(c) + n$$

$$\Rightarrow n \leq \log(c) \quad \forall n > n_0$$

Para $n = n_0 + \lceil c \rceil \quad n > \log(c) \text{ y } n \leq \log(c) \quad \square$

$$\therefore 2^{2^n} \neq O(2^n)$$

c) $3^n = O(2^{n^2})?$

Hint: Usar logaritmos

c) ¿ $3^n = \Theta(2^{n^2})$? Verdadero

$$3^n \leq C \cdot 2^{n^2} \text{ para } c > 0 \text{ y } \forall n > n_0 \in \mathbb{N}$$

Porque para $n \geq 2$ $3^n < 2^{n^2} \Rightarrow \exists c=1 \text{ y } n_0=2$
y $3^n \leq c \cdot 2^{n^2}$.

Sat:s factibilidad

1. Dar una codificación de fórmulas booleanas en FNC en el alfabeto {0, 1}

Se inicia la codificación con $1^n 01^m 0$, donde n indica el número de variables y m el número de cláusulas. A partir de ahí consideremos

- $1^k \equiv x_k$ p.a. $k \in \{1, 2, \dots, n\}$
- $0 \equiv \vee$
- $00 \equiv \neg \Rightarrow \vee \neg \equiv 000$
- $0000 \equiv) \wedge (\Rightarrow) \wedge (\neg \equiv 000000$
- $00000 \equiv (\Rightarrow (\neg \equiv 000000$
- $00000000 \equiv)$

Ejemplos

- $x_1 \equiv 1010000010000000$
- $x_1 \wedge x_2 \equiv 11011000000100001100000000$
- $x_1 \vee x_2 \equiv 1101000000101100000000$
- $x_1 \wedge \neg x_1 \equiv 101100000010000000100000000$
- $(x_1 \vee \neg x_2) \wedge x_3 \equiv 111011000000100011000011100000000$
- $\neg x_1 \equiv 101000000010000000$
- $x_1 \wedge (x_2 \vee x_3) \wedge (x_4 \vee \neg x_2) \equiv 111101110000001000001101110000
111100011000000000)$

En el peor de los casos se necesita $n+m+1+5+6 \cdot (m-1)(3n-1)+2+5+8 = n+m+21+6(3mn-m-3n+1) = 18mn-5m-18n+27 \therefore \Theta(m \cdot n)$

$$\approx \Theta(n^2)$$

Problema de Colouring

Dar algoritmos deterministas para 2-colouring y 3-colouring.

Analizarlos. ¿son de tiempo polinomial?

2-colouring

Dada una gráfica $G(V, E)$ se ordenan los vértices de forma arbitraria. Dados los símbolos 0 y 1 y su ordenamiento lexicográfico consideramos todos los elementos del $\underbrace{00\dots 0}_n$ al $\underbrace{11\dots 1}_n$ donde $n = |V|$.

Cada uno de los elementos del 0...0 al 1...1 corresponden a una posible coloración de 2 colores, donde el elemento i de cada cadena indica el color del vértice i ; el valor 0 indica el primer color y el valor 1 el segundo color. En cada iteración se evalúa la coloración comparando el color de cada vértice con el de sus vecinos si algún vértice tiene la misma coloración que sus vecinos, entonces la coloración falla y se prueba la siguiente. Si una coloración satisface las condiciones, entonces se acepta la coloración y el algoritmo se detiene. Si no acepta ninguna coloración, el algoritmo rechaza y se detiene. El algoritmo siempre se detiene.

Podemos escribir el algoritmo como:

Dada una entrada G (A matriz de adyacencia de G)

$O(n) n = \dim(A)[0]$

$O(n) \text{ conteo} = \underbrace{\overline{0, 0, \dots, 0}}_n$

Mientras $\text{conteo} \leq \underbrace{1, 1, \dots, 1}_n$ 2^n veces

Para i desde 0 hasta n n veces

$j = 0 ; \text{aux} = 1$ $O(1)$

Mientras $j < n$ y $\text{aux} = 1$ n veces

Si $A[i, j] == 1$ $O(1)$

Si $\text{conteo}[i] == \text{conteo}[j]$ $O(1)$

$\text{aux} = 0$ $O(1)$

$j = j + 1$ $O(1)$

Si $j == n$ y $\text{aux} == 1$ $O(1)$

regresar(conteo) $O(1)$

detener. $O(1)$

incrementar conteo (hardcodeado)

El algoritmo dado tiene 3 ciclos, uno para cada coloración, uno para evaluar cada vértice y otro para comparar con todos los vecinos. En el peor de los casos se evalúan las 2^n coloraciones para los n vértices contra los n vértices, por lo que es $O(n^2 2^n)$. Este algoritmo es de tiempo exponencial.

Para 3-colouring se tiene la misma idea, cambiando únicamente los símbolos a 0, 1, 2. Donde el símbolo 0 corresponde al primer color, el símbolo 1 al segundo color y el símbolo 2 al tercer color. Es con estos símbolos que se define un orden lexicográfico que va del $\underbrace{00\dots 0}_n$ al $\underbrace{22\dots 2}_n$

Del algoritmo anterior sólo cambia el ciclo externo que en vez de contener 2^n posibilidades, contiene 3^n posibilidades. Por lo que todo el algoritmo es $O(n^2 3^n)$, el cual también es exponencial.

Nota: Con algoritmos de programación dinámica es posible disminuir la complejidad. Además para 2-colouring se puede verificar si la gráfica es bipartita lo cual se puede hacer en tiempo lineal.

Knapsack

a) Dar un algoritmo determinista para knapsack

Se ordenan los objetos, se evalúan primero las posibilidades donde se selecciona el primer objeto, se van agregando objetos uno a uno, si: $\sum w_i \leq W$ y $\sum v_i \geq k$, para todo elemento i seleccionado, entonces se acepta y se regresa la selección de objetos. Si: $\sum w_i > W$ se elimina el último objeto seleccionado y se agregan los objetos siguientes. Si: se llega al último objeto y $\sum v_i < k$, se evalúan los casos donde no está el primer objeto, en este caso el 2ndo objeto se puede considerar como el primero y el proceso se repite. Este proceso equivale a recorrer un árbol binario.

con búsqueda de profundidad.

Dadas las entradas v, w, ω y K :

$$C = \{1\}, n = |v|, i = 1, sv = v_1, sw = w_1, j = 1 \quad O(1)$$

Mientras ($sv < k \& i < n$)

$$i \leftarrow i + 1$$

$O(1)$

$$S: ((sw + wi) < \omega)$$

$$sv = sv + v_i$$

$$sw = sw + w_i$$

$$C = C \cup \{i\}$$

$$Si \ sv \geq k$$

Acepta y se detiene

Si ($i == n$) \leftarrow i puede llegar hasta n

$$S: (|C| == n - j + 1 \circ j == n)$$

Rechaza y se detiene

$j = j + 1 \leftarrow j$ puede llegar hasta n

$$i = j$$

$$C = \{i\}, sv = \{v_i\}, sw = \{w_i\}$$

fin

$O(1)$

Hay un ciclo que depende de 2 variables, la i y la j , el resto de las instrucciones son $O(1)$. La variable j puede tomar valores del 1 al n una única vez en el peor de los casos. Mientras que i puede recorrer primero del 1 al n , luego del 2 al n , del 3 al n ... y el n .

Con esto último podemos saber que el ciclo principal se repite a lo más $\frac{n(n+1)}{2}$ veces lo que hace que este problema sea de $O(n^2)$, siempre y cuando se definen umbrales K y ω .

b) Dar un algoritmo no determinista para knapsack

Consideramos todas las posibles combinaciones de elementos, evaluamos no determinísticamente cada rama que contiene una posible combinación y verificamos si cumple con $\sum v_i \geq k$ y $\sum w_i \leq W$. Si alguna cumple ambas condiciones, entonces acepta, en caso contrario rechaza. En el caso no determinista el algoritmo es $O(n)$.

c) Dar cotas superiores asintóticas para los algoritmos de a) y b)

¿Son polinómicas?

Como vimos anteriormente el algoritmo en a) es $O(n^2)$ y el algoritmo en b) es $O(n)$.

MT de k-cintas a MT de una cinta

Tarea: Probar que si $L \subseteq \Sigma^*$ es aceptado por una MTD con k-cintas en tiempo $O(f(n))$ entonces L es aceptado por una MTD con una sola cinta en tiempo $O(f^2(n))$.

Dada M una MTD con k-cintas, se puede definir N una MTD de una sola cinta de la siguiente manera. Sea T el conjunto de símbolos de M, $\forall s_i \in T$ definimos un nuevo símbolo s'_i que indica que una de las cabezas lectoras de M está viendo el símbolo s_i . Llamemos T' al conjunto de símbolos s'_i . Ahora definimos $T_n = T \cup T'$ como el conjunto de símbolos de N. Y vamos a considerar que N tiene K carriles que pueden ser leídos por la cabeza lectora de N de forma simultánea.

Cada paso de la matriz M puede ser simulado de la siguiente manera: (siguiente página)

Dada una entrada x :

- $\Theta(k)$ • Definir el primer símbolo de la cinta en cada carril como $t' \in T'$
- $\Theta(n)$ • Escribir ax en el primer carril a la derecha de t'

Mientras no se acepte o rechace

- $\Theta(k \cdot f(n))$ • Recorrer la cinta de izquierda a derecha, para cada carril en la cinta verificar el símbolo si el carril i tiene un símbolo $s' \in T'$ entonces registramos que M está viendo s en la cinta k

- $\Theta(f(n))$ • Regresar la cabeza al inicio de la cinta

(hardcoded) • Evaluar un paso de M con $\delta_M: Q \times T^k \rightarrow Q \times T^k \times \{L, R\}^k$

$\Theta(1)$

- y conservar el nuevo estado, los valores de reescritura y los movimientos
- Si el estado al que se llega es de aceptación, entonces N también acepta y si es de rechazo N también rechaza.

k veces

- Para cada carril desde 1 hasta k .

- $\Theta(f(n))$ { • Encontrar un símbolo en T'
- Reescribirlo por el nuevo símbolo correspondiente
- Hacer el movimiento a la izquierda o derecha según el resultado de δ_M para la cinta que corresponde al carril en N .
- Leer el símbolo y cambiarlo por su equivalente en T' .
- Regresar la cabeza lectora al inicio.

Dada una entrada x la máquina N replica todas las transiciones de M .

y en caso de que M acepta a x , entonces N también acepta ax . Si:

N acepta a x , entonces es necesario que M haya aceptado a x . Sabemos

que M es $\Theta(f(n))$, entonces el número de iteraciones externas de N

está acotado por $\Theta(f(n))$. Los ciclos internos dependen de K y del

tamaño de la cinta. Como no conocemos más información de las

cotas espaciales de M , sólo podemos decir que el espacio está acotado.

Por $c \cdot f(n)$ y por lo tanto es $\Theta(f(n))$ y el tamaño de la cinta es $\Theta(f(n))$

→

entonces, los ciclos internos son $O(k \cdot f(n))$, como k es una constante en M, entonces los ciclos son $O(f(n))$. Considerando los ciclos internos ($O(f(n))$) y el ciclo externo $O(f(n))$, se tiene que la simulación de M con N es $O(f^2(n))$ □

Reachability

Probar que $\text{reachability} \in \text{TIME}(n^2)$ donde n es el # de vértices del grafo de entrada.

Definimos dos conjuntos, el C y el C', el C contiene todos los nodos por los que no se ha pasado y el C' todos los nodos por los que se ha pasado y en el orden correspondiente. C puede ser implementado con una pila. Se parte con $C = V - \{f\}$ y $C' = \{f\}$ donde f es el nodo fuente. Se busca el primer vecino de f se compara con el nodo destino d, si se trata de d se acepta, si no se busca el primer vecino del vecino de f distinto de f. Si no tuviera ningún vecino entonces se busca el segundo vecino de f, en caso que sí tenga un vecino se repite el proceso. El proceso se termina cuando se encuentra un camino hasta d o cuando $C' = \emptyset$.

Ver algoritmo en página siguiente

Dadas las entradas $G(V, \mathcal{E})$, f y d

$$C = V - \{f\}; \quad C' = \{f\}; \quad j = f \quad O(1)$$

Mientras ($C \neq \emptyset$ y $C' \neq \emptyset$) \leftarrow alguno se vuelve \emptyset en al menos n iteraciones

$$i = 1$$

Mientras $i \leq n$ Para cada iteración se repite al menos n veces

$$\text{Si } (i \in C)$$

$$\text{Si } (e_{ji} \in \mathcal{E}) \quad \begin{array}{l} \text{Si: se define una matriz de incidencia} \Rightarrow O(1) \\ \text{s: no} \\ \quad O(n). \end{array}$$

$$C = C - \{i\}$$

$$C' = C' \cup \{i\}$$

$$\text{Si: } (i == d)$$

| regresa C' y acepta.

Si: no

$$j = i$$

$$i = 0$$

$$i = i + 1$$

$O(1)$

$$\text{Si } i == n + 1$$

$$C' = C' - \{j\}$$

$$i = j + 1$$

$$\text{Si: } C' == \emptyset$$

| Rechaza

Si: no

| $j = \text{ultimo elemento agregado a } C'$

Si la verificación $e_{ji} \in \mathcal{E}$ se hace en $O(1)$, entonces el algoritmo depende de los ciclos los cuales son a lo más $O(n^2)$. Reachability se puede resolver en $O(n^2)$

Clases de Complejidad.

1. Probar que P es cerrada bajo una intersección de lenguajes.

Repetir para NP .

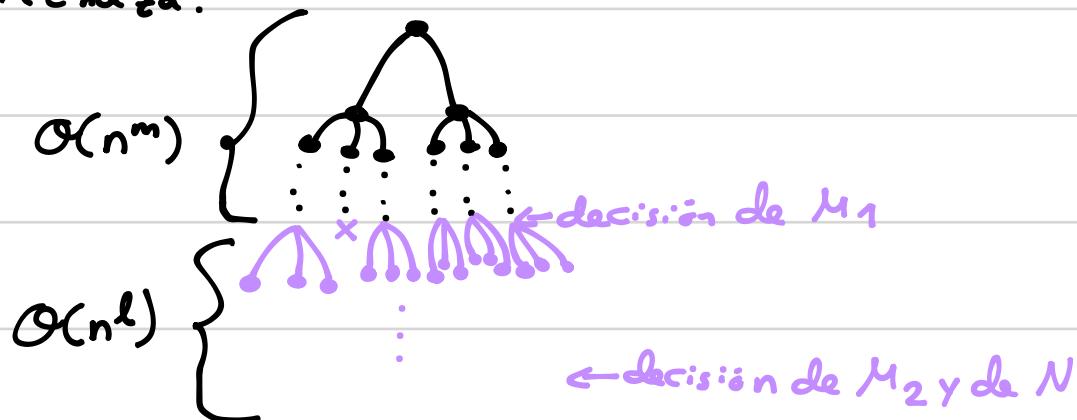
Dem P es cerrado bajo intersección de lenguajes.

Sean $L_1, L_2 \in P$, $\exists M_1, M_2$ MTD de k_1, k_2 cintas tales que M_1 reconoce $a x \Leftrightarrow x \in L_1$ y M_2 reconoce $a x \Leftrightarrow x \in L_2$.

Definimos N MTD de $\max(k_1, k_2)$ cintas. Supongamos que $k_1 = \max(k_1, k_2)$, entonces definimos M_2' con k_1 cintas, en que la cabeza lectora de las últimas $(k_1 - k_2)$ cintas replican el movimiento de la primera y si siempre escribe el símbolo que representa al vacío. A partir de M_1 y M_2' se construye N con $k = \max(k_1, k_2)$ cintas, de la siguiente manera. Se inicializa N escribiendo $a x$ en la primera cinta y se hacen las transiciones de M_1 hasta llegar a un estado de aceptación o rechazo. Si M_1 rechaza, entonces N rechaza. Si M_1 acepta, entonces se borra el contenido de las cintas, se escribe $a x$ en la primera cinta y se hacen las transiciones de M_2' . Si M_2' rechaza, entonces N rechaza y si M_2' acepta, entonces N acepta. Como $M_1 \in P$, entonces $\exists l \in \mathbb{N} \text{ s.t. } M_1 \in O(n^l)$, por lo que simular M_1 toma tiempo $O(n^l)$, el borrar el contenido de las cintas es a lo más $O(n^l)$. M_2' al igual que M_2 está en P $\therefore \exists m \in \mathbb{N} \text{ s.t. } M_2' \in O(n^m)$. Entonces N es $O(n^{\max(l, m)})$, por lo que N reconoce a L en tiempo polinomial. Además L es reconocido por $N \Leftrightarrow L$ es reconocido por L_1 y por $L_2 \Leftrightarrow L \in L_1 \cap L_2$ $\therefore N$ reconoce a la intersección de L_1 y L_2 en tiempo polinomial. $\therefore P$ es cerrado bajo la intersección.

Dem: NP es cerrado bajo intersecciones.

Sean $L_1, L_2 \in NP$, entonces $\exists M_1, M_2$ MTND tales que M_1 reconoce a L_1 en tiempo $O(n^m)$ y M_2 reconoce a L_2 en tiempo $O(n^l)$ con $l, m \in \mathbb{N}$. Podemos definir a N una MTND de la siguiente forma. N hace una adivinación para M_1 , si M_1 rechaza, entonces N rechaza. Si no para cada rama donde M_1 acepta se hace una adivinación sobre M_2 y si M_2 acepta, entonces N acepta, si no N rechaza.



N decide un lenguaje L en $O(n^l + n^m) \approx O(n^{\max(l, m)})$, por lo cual N es una MTND que decide en tiempo polinomial. Además L es reconocido por $N \Leftrightarrow M_1$ reconoce a L y M_2 reconoce a $L \Leftrightarrow L \in L_1 \cap L_2$

$$\therefore NP \text{ es cerrado bajo la intersección.}$$

2. La estrella de Kleene de $L \subseteq \Sigma^*$ se define como

$$L^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ y } x_i \in L\}$$

Muestra que

a) NP es cerrado bajo estrella de Kleene.

$$S: x \in L^* \Rightarrow x = x_1 x_2 \dots x_k = x_1^{n_1} x_2^{n_2} \dots x_1^{n_k} x_2^{n_k} \dots x_1^{n_k} x_2^{n_k} \dots x_k^{n_k}$$

Para $k \in \mathbb{N}$ y $n_i \in \mathbb{N} \forall i \in \{1, \dots, k\}$. Sin embargo dada una entrada x no sabemos el valor de k ni de n_i . Únicamente sabemos que $k \leq |x| = n$.

Por lo tanto para probar que una entrada $x \in L^*$ es necesario considerar las posibles agrupaciones de símbolos de x y verificar si dada una forma de agrupar los símbolos de x resulta que cada grupo de símbolos está en L . Para lograr reconocer una cadena en L^* definimos la siguiente

MTND, M.

Dada una entrada x :

Mientras $x \neq \lambda$ A lo más $|x|=n$ veces.

Adivinar no determinísticamente la primera agrupación de x como

x_0 y redefinir x como la cadena de símbolos que siguen a x_0

$\Theta(|x_0|^m)$ Si $x_0 \notin L$

rechazar la rama correspondiente y terminar el ciclo

Si para alguna adivinación $x = \lambda$ y no hubo rechazo entonces

M acepta.

La rama más larga es aquella donde cada símbolo en la cadena se verifica como elemento de L , pero en este $\Theta(|x_0|^m) = \Theta(1^m) = \Theta(1)$. En el caso de la rama más corta, en que todo x es la primera cadena que se evalúa si pertenece a L . En este caso la evaluación es $\Theta(n^m) = \Theta(n^m)$. Con esto tenemos que la complejidad de M depende de la máquina que decide a L en $\Theta(n^m)$ y el tiempo de M es también $\Theta(n^m)$.

∴ NP es cerrado bajo la estrella de Kleene. \blacksquare

Reducciones

1. Probar que si $A \leq_p B$ y $B \in P \Rightarrow A \in P$.

$A \leq_p B \Rightarrow \exists f: \Sigma^* \rightarrow \Sigma^*$ f se calcula en tiempo polinomial y

$x \in A \Leftrightarrow f(x) \in B$. Como $B \in P \exists M$ una MTD con k -cintas que decide a B en tiempo polinomial, podemos definir a N una MTD con k' -cintas con $k' = \max(k, k_f)$ con k_f el número de cintas de la MTD que calcula la función f .

Para una entrada $x \in \Sigma^*$, N primero calcula $f(x)$ y N decide a $f(x)$ a partir de la máquina M. Ahora probamos que N decide a A. \rightarrow

Si $x \in A \Rightarrow f(x) \in B$, como M decide a B , N acepta a x .

Si N acepta a $x \Rightarrow M$ acepta a $f(x) \Rightarrow f(x) \in B$, como f es una reducción de A en B , $\Rightarrow x \in A \therefore N$ acepta a $x \Leftrightarrow x \in A \therefore N$ decide a A .

Solo falta mostrar que N decide a A en tiempo polinomial. Como $A \leq_p B$, $f \in O(n^l)$ para alguna $l \in \mathbb{N}$ y como $B \in P$, M decide a B en tiempo $O(n^m)$ para alguna $m \in \mathbb{N} \Rightarrow N$ decide a A en tiempo $O(n^l + n^m) = O(n^{\max(l, m)})$ como $\max(l, m) \in \mathbb{N}$, N decide a A en tiempo polinomial $\therefore A \in P \quad \square$

Independent Set

Independent SET $\in NP$ (\exists MMND en tiempo polinomial que lo resuelve)

Se define una máquina M que hace lo siguiente:

Dadas las entradas G y k :

Se hace una adivinación en la que se seleccionan k vértices de G .

Si para cualesquiera 2 vértices v_i, v_j en la adivinación existe una arista e_{ij} en G que conecta a v_i y a v_j entonces se rechaza. Si no M acepta.

M acepta si al menos una rama acepta. El proceso de verifcar si $\exists e_{ij}$ que conecta a v_i y v_j es $O(k^2)$ si G está en términos de la matriz de adyacencia y el acceder a la entrada $A[i, j]$ es $O(1)$. El hacer la adivinación sería $O(1)$ y asr mismo el rechazar o aceptar es $O(1)$. Por lo tanto, la máquina M decide en tiempo $O(k^2)$, en el peor de los casos $k = n \therefore M$ es $O(n^2)$. Entonces M

decide independent set en tiempo polinomial y como es no determinista, entonces Independent Set $\in \text{NP}$. \square

CLIQUE y NODE COVER

Probar que CLIQUE y NODE COVER $\in \text{NP}$.

a) CLIQUE = $\{(G, k) \mid G \text{ tiene un clique de tamaño } k\}$ $\in \text{NP}$.

Sea M una MTND tal que dada G en términos de su matriz de adyacencia A y $k \in \mathbb{N}$

Si $k > |G|$

rechaza

Si no se hace una adivinación en la que se eligen k vértices y se guardan en un conjunto V'

Para cada $v_i \in V'$

Para cada $v_j \in V'$

Si $v_i \neq v_j$ $O(1)$

Si $A[v_i, v_j] = 0$ $O(1)$

rechaza $O(1)$

Acepta $O(1)$

$O(n^2)$

Si alguna rama acepta, entonces M acepta

Si $(G, k) \in \text{Clique} \Rightarrow \exists C \subseteq V \text{ s.t. } |C|=k$ y

$\forall i \neq j \in C \exists e: v_i, v_j \in E$. Entonces dada la adivinación que corresponde a C , M acepta, ya que todos los vértices en C están conectados por una arista y esto se puede comprobar con la matriz

de adyacencia en que $A[v_i, v_j] = 1$ si $\exists e \in v_i \in \Sigma$ arista que conecta a v_i y v_j . Como C es un clique $A[v_i, v_j] = 1$ en todos los casos \therefore Esa rama va a aceptar y M acepta.

Si M acepta, entonces existe un subconjunto V' de V con k elementos para los cuales $A[v_i, v_j] \neq 0$ para $v_i, v_j \in V'$ con $i \neq j$. Esto indica que para cualesquier dos vértices en V' existe una arista que los une y por lo tanto se tiene un grafo completo $\therefore V'$ es un clique con k elementos de G .

Con esto tenemos que M reconoce a CLIQUE. Además cada rama de M utiliza un tiempo $O(n^2)$ para llegar a un estado de aceptación o rechazo en el peor de los casos. $\therefore M$ es MTND y termina en tiempo polinomial \therefore

CLIQUE $\in NP \quad \square$

b) Nodos Cover = $\{ (G, k) \mid G$ tiene una cubierta de nodos de tamaño k o menor $\}$.

Definimos M MTND tal que dada la matriz de adyacencia de G y $k \in \mathbb{N}$, $k \leq |V|$

Se hace una adyacencia en la que se eligen k vértices de G y se guardan en V'

Para i desde 1 hasta $|V|$

Para j desde 1 hasta $|V|$

Si $A[i, j] = 1$

$B = 0, l = 1$

Mientras $l \leq k$, $B = 0$

Si $V'[l] = i$ o $V'[l] = j$

$B = 1$

Si $B = 0$

rechaza

$O(k \cdot n^2)$

$O(l)$

Acepta

Si alguna rama acepta, M acepta

Si G tiene un nodo cover de tamaño k , para la adivinación correspondiente N , se tiene que para cualquier arista $e_{ij} \in \Sigma$ $\exists v_i \in N$ tal que $v_i = i$ ó $v_i = j$. \Rightarrow Cada vez que se tenga un valor distinto de 0 en la matriz de adyacencia va a existir un elemento en N que coincida con i o j , por lo que dentro del ciclo de M no se va rechazar y al terminar el ciclo se llegará a un estado de aceptación. $\therefore S: (G, k) \in \text{Node Cover} \Rightarrow M \text{ acepta a } (G, k)$.

Si M acepta una entrada $(G, k) \Rightarrow \exists$ una adivinación V' tal que para cualquier vértice $e_{ij} \in \Sigma$ representado en la matriz de adyacencia como $A[i, j] = 1$ se tiene que algún elemento en $v_l \in V'$ es tal que $v_l = i$ ó $v_l = j$ con lo que se cumple que V' es una cubierta por nodos de tamaño k . $\therefore M$ reconoce a node cover.

Para cada rama M tarda $\Theta(kn^2)$ $\therefore M$ es $\Theta(n^2)$, lo cual es tiempo polinomial. Como $\exists M$ MTND que reconoce a NODE COVER en tiempo polinomial. $\text{NODE COVER} \in \text{NP}$.



Lema

- a) Independent set \leq_p CLIQUE
- b) Independent Set \leq_p NODE COVER

a) Mostrar que $\exists f: \Sigma^* \rightarrow \Sigma^*$ $f \in \Theta(n^k)$ y $x \in \text{Independent set} \Leftrightarrow f(x) \in \text{NODE COVER}$

Definimos f de la siguiente manera, dada una gráfica G en términos de su matriz de adyacencia:



Dada la matriz de adyacencia de G , k y $n = |V|$

Para i desde 1 hasta n

 Para j desde 1 hasta n

 Si $A[i, j] = 0$

 | $A[i, j] = 1$

 S: no

 | $A[i, j] = 0$

} $\mathcal{O}(n^2)$

Regresar (A, k)

↑ gráfica definida a través de A , matriz de adyacencia

La función f termina en $\mathcal{O}(n^2)$ $\therefore f$ es polinomial. Ahora debemos mostrar que $x \in IS \Leftrightarrow f(x) \in NC$

Si $(G, k) \in IS \Rightarrow \exists I \nsubseteq |I|=k \subset I$ es un conjunto independiente. $f(G, k)$ regresa una pareja (A, k) donde k no cambia y A es la matriz de adyacencia de G^c , por la proposición se tiene que I es solución de CLIQUE de G^c , como $|I|=k \Rightarrow (A, k) \in \text{CLIQUE de } G^c$.

Si $f(x) = (A, k) \in \text{CLIQUE de } A \Rightarrow \exists C \subseteq V \nsubseteq |C|=k$ y C es un clique de A , sabemos que A es la matriz de adyacencia de una gráfica $G = (G^c)^c$, sea $G' = G^c \Rightarrow$ la gráfica G' tiene a C como independent set. Como $|C|=k \Rightarrow G'$ tiene un conjunto independiente de tamaño k . $\therefore (G', k) \in \text{independent set}$.

Como f es polinomial y se cumple que $x \in IS \Leftrightarrow f(x) \in \text{CLIQUE}$ $\Rightarrow f$ es una reducción polinomial de IS a CLIQUE $\Rightarrow IS \leq_p \text{CLIQUE}$ \square

b) Independent SET \leq_p NODE COVER

Sea $f: \Sigma^* \rightarrow \Sigma^*$ definida como sigue

Dada una gráfica G y $k \in \mathbb{N}$

$$n = |V| \quad O(n) \quad (\cup O(1))$$

$$k = n - k \quad O(1)$$

Regresar $(G, n - k)$

La función f es $O(n)$ ∴ es polinomial. Si $(G, k) \in IS$
 $\Rightarrow \exists I \subseteq V$ conjunto independiente de G s.t. $|I| = k$ por la
proposición $V - I$ es una cubierta por nodos de G y $|V - I| =$
 $|V| - |I| = n - k$, ya que V e I son finitos. ∴ G tiene una
cubierta por nodos de tamaño $n - k$ ∴ se cumple que $f(G, k) =$
 $(G, n - k) \in NODE COVER$.

Si $f(x) = (G, k') \in NODE COVER \Rightarrow \exists N$ s.t. $|N| = k'$ y
es una cubierta por nodos de G , por la proposición $N = V - I$
con I conjunto independiente. ∴ G tiene un conjunto
independiente I , s.t. $|I| = n - k' = k$ ∴ $x = (G, k) \in IS$
 $\therefore \exists f$ polinomial tal que reduce IS a $NODE COVER$
 $\therefore IS \leq_p NODE COVER \quad \square$

Complejidad

1. Demostrar que P es cerrado bajo reducciones.

Sean L y $L' \in P$ y $L \leq_p L' \Rightarrow \exists f: \Sigma^* \rightarrow \Sigma^*$ computable
en tiempo polinomial s.t. $x \in L \Leftrightarrow f(x) \in L'$. Como $L' \in P$

$\exists M$ una MTD tal que decide a L' en tiempo polinomial.

\Rightarrow Definimos N una MTD tal que recibe una entrada x ,
la aplica la función f con lo que se obtiene $f(x)$ y simula
 M con entrada $f(x)$ y N acepta si M acepta y rechaza e.o.c.
Se tiene que N tarda $O(|x|^k) + O(|f(x)|^m)$. Como

$f(x)$ es polinomial $\exists l \in \mathbb{N} \text{ s.t. } f(x) \text{ se logra en tiempo } O(n^l)$ con $n = |x|$, además $|f(x)| \leq c \cdot n^l$, para alguna $c \in \mathbb{R}^+$ fija, ya que en tiempo $O(n^l)$ no es posible escribir más símbolos que el número de posos.

$\therefore O(|f(x)|^m) \in O((n^l)^m) = O(n^{lm})$ como $l, m \in \mathbb{N}$
 $l, m \in \mathbb{N} \therefore$ Simular M con entrada $f(x)$ es $O(n^q)$ con $q = lm \in \mathbb{N}$. \therefore La máquina N toma $O(n^k) + O(n^q)$
= $O(n^k + n^q)$. Sea $r = \max(k, q) \in \mathbb{N}$ se tiene que N se calcula en $O(n^r)$ con $r = \max(k, q) \in \mathbb{N}$. $\therefore N$ es computable y termina en tiempo polinomial. Ahora probamos que N decide a L .

Sea $x \in L \Rightarrow f(x) \in L'$ como M acepta a $f(x) \in L' \Rightarrow N$ acepta.

Si N acepta a $x \Rightarrow M$ acepta a $f(x) \Rightarrow f(x) \in L' \Rightarrow x \in L$

$\therefore N$ decide a L en tiempo polinomial

$\therefore L \in P$

$\therefore P$ es cerrado bajo reducciones \square