Notas del Curso de Complejidad 2021-1

1 Máquinas de Turing

1.1 Introducción

En 1906 se declara el Problema 10 de Hilbert: ¿Existe un algoritmo que decida si una ecuación diofantina dada (con cualquier mínimo de variables) tiene solución (entera)?
En 1928 se genera el problema de Hilbert-Ackermann : ¿Hay un algoritmo que dado un enunciado, un conjunto de axiomas y un conjunto de reglas de inferencias, decida si el enunciado se puede deducir a partir de los axiomas mediante las reglas de inferencia?
Leibniz propuso el tener "un tipo de álgebra en el que todas las verdades de la razón pudieran reducirse a algún tipo de cálculo"

Un problema de decisión P lo podemos pensar como $f : A \to \{0, 1\}$ donde A es el conjunto de instancias (o entradas) de P. $f^y[0]$ conjunto de no-instancias $f^y[1]$ conjunto de sí.instancias

1.2 Máquina de Turing

Def Una Máquina de Turing es una noneta ordenada $M = (Q, \Sigma, \Gamma, \sqcup, \vdash, \delta, q_0, q_a, q_r)$

- Q es un conjunto no vacío (de entrada)
- Σ es un conjunto no vacío (alfabeto de entrada)
- Γ es un conjunto no vacío (alfabeto de cinta)
- $\delta : Q' \times \Gamma \to Q \times \{L, R\}$ donde $Q' = Q - \{q_0, q_r\}$
- $q_0 \in Q$ (estado inicial)
- $q_a \in Q$ (estado aceptación) Estado de paro
- $q_r \in Q$ (estado rechazo) Estado de paro
- $\sqcup \in \Gamma - \Sigma$ (símbolo en blanco)
- $\vdash \in \Gamma - \Sigma$ (marcador o delimitador izquierdo de inicio de cinta)

Para cualquier $q \in Q$ existe $p \in Q$ tal que $\delta(q, \vdash) = (p, \vdash, R)$

Una configuración de la máquina de Turing se determina por:

- Estado q
- Posición de la cabeza
- Contenido de la cinta uv

La configuración se ve como uqv, q es el cabezal que se encuentra en la primera posición de v.
Ejemplo: Si M tiene entrada w su posición inicial es $q_0 \vdash w$
La configuración α da lugar a la configuración β $\alpha \to \beta$ y se define como sigue:

- $\alpha \to^0 \alpha$
- Diremos que $\alpha \to^{n+1} \beta$ si existe una configuración Γ tal que $\alpha \to^n \Gamma$ y $\Gamma \to \beta$

- $\alpha \to^* \beta$ si $\alpha \to^n \beta$ para algún $n \in \mathbb{N}$

Una configuración e aceptación (o rechazo) es cualquiera en la que el estado sea q_a (ó q_r). Éstas configuraciones son de paro.

Si T es una MT y $w \in \Sigma^*$ se dice que T acepta w si $q_0 \to^n xq_ay$ para cadenas $x, y \in \Sigma^*$.
El lenguaje de T, L(T) es el conjunto $L(T) = \{x \in \Sigma^* | T \text{ acepta a } x\}$.
Si $L \subseteq \Sigma^*$ es un lenguaje, se dice que T acepta a L si L(T) = L.
Los lenguajes regulares pueden ser reconocidos por un autómata finito determinista
Algoritmo Algoritmo para convertir de DFA (autómata finito determinista) a TM (máquina de Turing)

1. Copiar cada estado
2. Cambiar cada transición con etiqueta $'x'$ por una con etiqueta $'x/x, R'$
3. Agregar un estado de aceptación
4. Para cada estado final en el autómata, agregarle una transición $\sqcup/\sqcup, R$

¿Cómo describimos las máquinas de Turing? Con pseudocódigo Ejemplo: Describimos la máquina de Turing que obtiene las potencias de dos como sigue: M = "Con entrada $\vdash w$:

1. Recorre la cinta de izquierda a derecha, tachando un cero no y uno si
2. Si en 1 la cinta contenía un único cero, *acepta*
3. Si en 1 la cinta contenía más de un creo, y un número impar de ceros, entonces *rechaza*
4. Regresa el cabezal a extremo izquierdo de la cinta
5. Va a 1"

Las máquinas de Turing pueden computar funciones parciales.
Tenemos $f : D \to C$ $D \subseteq \Sigma^*$ $C \subseteq \Gamma^*$.
Def. Sea R una MT, k un natural y f una función parcial sobre $(\Sigma^*)^k$ con valor en Γ^*. Decimos que T computa a f si para cada $(x_1, ..., x_k)$ en el dominio de f, $q_0 \vdash x_1 \sqcup x_2 \sqcup ... \sqcup x_k \to^k q_0 \vdash f(x_1, ..., x_k)$ y ninguna otra entrada que sea una k-ada de cadenas es aceptada por T.
Una función parcial $f : (\Sigma^*)^k \to \Gamma^*$ es Turing-computable (computable) si hay una máquina de Turing que la computa.
Al fijar el contradominio y la aridad de la función, entonces una MT calcula una única función parcial.

Una tercera opción del cabezal de la MT es quedarse en su lugar después de hacer algo (S). $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$.
M_1 tiene "la opción" S M_2 no tiene "la opción" S pero reconoce el mismo lenguaje que M_1 o computa la misma función.
Simulo una máquina de Turing que tiene la opción S con una máquina de Turing que no la tiene al poner un estado adicional, donde en las transiciones se escribe lo mismo que lea y se mueva a la derecha, luego se escriba lo mismo que lea y se mueva a la izquierda.

Algunas MT útiles que regresan el cabezal a donde estaba originalmente:

- NB (Next Blank)
- PB (Previous Blank)

- Copy ($\vdash x \to x \sqcup x$)
- Insert(σ)
- "Borrar la cinta" (Ponemos un delimitador derecho y borramos todo a su izquierda)
- Compare ($\vdash x \sqcup y$ acepta si $x = y$ o rechaza si $x \neq y$)
- R (Reverse)

Copy → NB → R → PB → Equal (Esta máquina de Turing reconoce palíndromos)

1.3 Máquinas de Turing multicintas

Una MT multicinta es como una MT pero tiene k cintas de memoria. Su definición solo difiere respecto a una MT usual en la función de transición $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$. $\delta(q_1, a_1, ..., a_k) = (q_1, b_1, ..., b_k, D_1, ..., D_k)$ $D_i \in \{L, R, S\}$.
Las máquinas de Turing multicinta no tienen más poder que las de una cinta. Para simularlas en una máquina de Turing de una cinta podemos usar bloques como letras de alfabeto y manejamos la posición del cabezal como un puntito en las distintas letras. Le hace una pasada para ver los cabezales y luego realiza otra para hacer las transiciones. La memoria está en la estructura de los estados (el control finito). Otra manera es que los espacios pares simulen una cinta y los impares otra (es trivial generalizarlo a k). Pero usaremos la siguiente:
Teorema Toda MT multicinta tiene una MT de una sola cinta equivalente.
Dem: Construimos, dada una MT multicinta M a la MT de una sola cinta S t.q. tiene los contenidos de las cintas de M en su cinta, separados por un símbolo especial #, y usando versiones especiales de cada símbolo en Γ_M para denotar donde están los cabezales $\dot{o}$
S = Con entrada $w_1, ..., w_n$

1. Poner su cinta en el formato que representa las k cintas. La cinta formateada se ve como $| \vdash |w_1|...|w_n|\#| \sqcup |\#| \sqcup |...|$.
2. Para simular un movimiento, S recorre la cinta desde $\vdash$ donde hasta el último #, para determinar los símbolos que están leyendo los k cabezales. Después S hace una segunda pasada para actualizar las cintas de acuerdo a la función de transición de M.
3. Si en cualquier punto, S mueve alguna de las cabezas virtuales hacia la almohadilla (#) de la derecha, en M entonces el cabezal correspondiente se movió a un espacio en blanco. Luego, basta con que se inserte un símbolo blanco en esa posición.

Corolario Si L es un lenguaje, entonces L es Turing-reconocible si y sólo si L es reconocido por alguna MT multicinta.
Dem: Toda MT es una MT multicinta.
La otra implicación es el teorema anterior ■

1.4 Máquinas de Turing no deterministas

Se diferencian de las deterministas está en que tenemos una nueva función de transición:
$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R, S\})$
Ya no es un algoritmo, sino que ahora la máquina "adivina" qué transición tomar para llegar al resultado. Se dice que acepta si al menos un camino lleva a aceptación. Se parecen a los autómatas no-deterministas. No son implementables en la vida real. Son equivalentes a una MT (usando una MT que haga BFS exhaustivo sobre el árbol de posibles configuraciones) Es importante pensar en términos de que la máquina "adivina" la respuesta correcta (SOLO si existe). Toda MT es MTN.

Ejemplo: Un número en unario ¿Es compuesto? Aplica la siguiente combinación de MTs NB → G (Guess) → NB → G → PB → M (Multiply) → PB → Equal
¿Para qué sirven las MTN? Estas máquinas teóricas realizan búsquedas exhaustivas "rápidamente" Resuelven problemas lentos muy rápido. "Si tuviéramos una MTN, podríamos resolver un montón de problemas que nos causan dolores de cabeza en la vida real" Como por ejemplo factorización de números y polinomios, agente viajero, SAT, etc.

Teorema Toda MTN tiene una MTD equivalente.
Dem: Sea N una MTN, proponemos a M una MTD de 3 cintas.

- En la primera cinta está la entrada y nunca se modifica.
- La segunda cinta es una copia de la cinta de N y se usa para simular a N
- La tercera cinta contiene las direcciones de los nodos del árbol de ejecución de N codificadas con $\Sigma = \{1, ..., b\}$ Existen direcciones inválidas.

M = "con entrada w

1. Copia la cinta 1 a la 2
2. Usa a la cinta 2 para simular a N con entrada w en una rama de su cómputo no determinista. Antes de cada transición, consulta la cinta 3 para determinar que decisión tomar entre las distintas posibilidades no deterministas. Si no hay más símbolos en la cinta 3 o se llega a una dirección inválida, aborta y va a 3. Si encuentra una configuración de rechazo va a 3. Si encuentra una configuración de aceptación, *acepta*
3. Reemplaza la cadena en la cinta 3 por la siguiente en el orden canónico de $\{1, .., b\}^*$
4. Va a 1"

Si una MTN M se ejecuta hay 3 posibilidades:

1. acepta Configuración de paro
2. rechaza Configuración de paro
3. se cicla

Corolario Un lenguaje es reconocible si y sólo si alguna MTN lo reconoce.
Una MT que no se cicla es un decididor.
Un lenguaje L es Turing-decidible si existe un decididor que lo reconoce.
Corolario Un lenguaje es decidible si sólo si alguna MTN lo decide.
Dem. La ida es trivial. Para el regreso notemos que cada longitud de cadena es un nivel del árbol, así que mantenemos un booleano que es v si todas las ejecuciones han terminado (en la longitud actual) f e.o.c. Cuando acaben las cadenas de longitud k, rechazamos si el booleano es v. En otro caso, vamos a la longitud k+1.

1.5 Enumeradores

Sea E una MT con k cintas $k \in \mathbb{Z}$ y sea $L \subseteq \Sigma^*$. Decimos que E *enumera* a L si:

1. El cabezal de la primera cinta nunca se mueve a la izquierda, nunca escribe un espacio en blanco y nunca modifica un símbolo que ya escribió.

2. Para cualquier $x \in L$ hay algún punto de la operación de E en la que la cinta 1 contiene $x_1\#x_2\#...\#x_n\#x$ donde $x_i \in L$ p.c $1 \leq i \leq n$ y $x_1, ..., x_n, x$ son todas distintas. Si L es finito no se imprimen símbolos después de la última #.

El lenguaje de E, es L, i.e. L(E) = L.

Teorema: Sea $L \subseteq \Sigma^*$ un lenguaje. Entonces L es reconocible si y sólo si es enumerado por algún enumerador; además L es decidible si y sólo si es enumerado en orden canónico por algún enumerador.
Demostración: Supongamos primero que L es decidible y sea M una MT que lo decide. Proponemos a E dado por
E ="Ignora la entrada

1. Genera la siguiente cadena w en orden canónico de Σ^*
2. Ejecuta M con entrada w e imprime a w si M acepta".

Como M es un decididor para L, E va a recorrer todas las cadenas en Σ^* y, en orden canónico, va a imprimir solamente las que están en L.
Supongamos que L es reconocido por M. Proponemos al enumerador E dado por: E ="Ignora la entrada

1. Para cada $i \in \mathbb{N}$
2. Genera la siguiente cadena si en orden canónico y le agrega a la lista $s_0, s_1, ..., s_i$
3. Ejecuta M por i pasos con cada una de las entradas s_0 hasta s_i
4. Si M acepta alguna de las entradas y no la había aceptado en una ejecución anterior, la imprime."

Si tenemos s_i y s_k tal que $i < k$,
Supongamos que E enumera a L. Proponemos a M para recorrer a L dado por: M="Con entrada w:

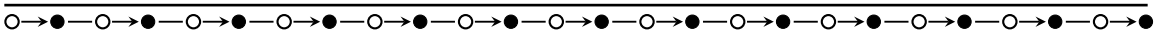
1. Ejecuta E. Cada vez que se imprima una cadena, la compara con w.
2. Si w se imprime en algún momento *acepta*."

Supongamos que E enumera a L en orden canónico, Proponemos a M dado por M="Con entrada w:

1. Ejecuta E. Cada vez que se imprima una cadena, la compara con w.
2. Si w se imprime en algún momento *acepta*.
3. Si la cadena impresa por E supera a w en orden canónico, *rechaza*."■

1.6 Tésis Church-Turing

Turing, Church, Goëdel et al formalizaron el concepto de algoritmo.
Los algoritmos desde el punto de vista intuitivo son equivalentes a las Máquinas de Turing.
Ejemplo: I : G es gráfica
Q: ¿Es G conexa?
M= "Con entrada $< G >$ con G una gráfica:



1. Elige el primer vértice de G y lo marca.
2. Repite el siguiente paso hasta que no se marquen vértices nuevos.
3. Para cada vértice en G, éste es marcado si es adyacente a algún vértice marcado.
4. Revisa si todos los vértices están marcados. En tal caso *acepta*, si no *rechaza*".

M decide A

1.7 Máquina Universal de Turing

U: "Con entrada $< M, w >$ con M una MT

1. Simula a M con entrada a w
2. *Acepta* si M acepta, *rechaza* si M rechaza.

2 Problema de aceptación

$A_{AFD} = \{< A, w > |$El AFD A acepta a la cadena w$\}$
Problema A_{AFD}
I: Un AFD A y una cadena w
Q: ¿A acepta a w?

Prop: A_{AFD} es decidible.
Dem: Proponemos una MT M que decida a A_{AFD}
M = Con entrada $< A, w >$ con a un AFD y w una cadena:

1. Simula A con entrada w.
2. Si la simulación termina en un estado de aceptación, *acepta*. Si no, *rechaza*■

Prop: A_{AFN} es decidible
Dem: Proponemos una MTD N que decida a A_{AFN}
N = "Con entrada $< B, w >$ con B un AFN y w una cadena:

1. Construye una AFD A equivalente a B.
2. Ejecuta M con entrada w.
3. *Acepta* si M acepta y *rechaza* si M rechaza ".

$A_{REX} = \{< R, w > |$R es una expresion regular que genera a w$\}$
Prop: A_{REX} es decidible
Dem: P = "Con entrada $< R, w >$ donde R es una regex y w una cadena:

1. Construye un AFN A equivalente a R.
2. Ejecuta N con entrada $< A, w >$
3. Responde lo mismo que N".

$E_{AFD} = \{< A > |$A es un AFD con L(A) = $\emptyset\}$
Prop: E_{AFD} es decidible
Dem: T = "Con entrada $< A >$ donde A es un AFD:

1. Marca el estado inicial de A.
2. Repite hasta que no se marquen estados nuevos
3. Marca todos los estados que tengan transición desde un estado marcado.
4. Si algún estado de aceptación está marcado, <u>*rechaza*</u>. Si no, <u>*acepta*</u>.

$EQ_{AFD} = \{< A, B > |$A y B son AFD tales que L(A)=L(B)$\}$
Prop: $C, L(C) = L(A) \triangle L(B)$
Dem: F = "Con entrada $< A.B >$ donde A y B es un AFD:

1. Construye a C tal que $L(C) = L(A) \triangle L(B)$
2. Ejecuta a T con entrada C
3. Si T acepta, <u>*acepta*</u>. Si no, <u>*rechaza*</u>■.

Función característica: El conjunto $S \subseteq A$ $\chi_S : A \to \{0,1\}$ $\chi_S(a) = 1$ si $a \in S$ 0 si $a \notin S$
$\omega = \{0, 1, 2, 3, ...\} = \mathbb{N}$ $(0, 1, 0, 1, ..)$ sería el conjunto de los impares.
$\mathcal{P}(\mathbb{N}) \sim^{\mathbb{N}} 2 = \{f : \mathbb{N} \to \{0, 1\}$
$\varphi : P(\mathbb{N}) \to^{\mathbb{N}} 2$ $A \mapsto \chi_A$ $\psi : 2 \to P(\mathbb{N})$ $f \mapsto \{i \in \mathbb{N} | f(i) = 1\}$
$\psi \cdot \varphi = 1_{P(\mathbb{N})}$ $\varphi \cdot \psi = 1_{P(\mathbb{N})}$
¿La potencia de los naturales es biyectable con las sucesiones binarias? $\mathbb{N} \sim \mathcal{P}(\mathbb{N})$ ¿Los naturales y la potencia de los naturales no son biyectables? $\mathbb{N} \sim \mathbb{N}_2$
Si fueran biyectables podrías tener: $f : \mathbb{N} \to^{\mathbb{N}} 2$ Si f fuera suprayectiva entonces toda sucesión binaria aparecería en la lista (sería imagen de alguien).
$g : \mathbb{N} \to 2$ $g(i) = 1 - f_i(i)$ $g = (1, 0, 1, 1, ...)$
Podemos llegar a contradicción porque: $\exists n \in \mathbb{N}$ $f_n(n) \neq g(n) = 1 - f_n(n)$! Por lo tanto, $\mathbb{N} \nsim \mathcal{P}(\mathbb{N})$
$\mathbb{N} \nsim \mathbb{N}_2$
"Nadie podrá expulsarnos de paraíso que Cantor ha creado"-Hilbert.

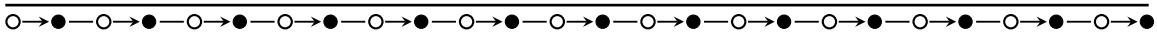
Prop. Existen lenguajes no reconocibles.
Dem: Si Σ es contable, entonces Σ^* es numerable. Sea $\mathcal{M} = \{< M > | Mesuna MT\} \subset Sigma^*$
$|\mathcal{M} \leq \aleph_0$
Si $\mathcal{L}L$ es el conjunto de todos los lenguajes sobre el alfabeto Σ, afirmamos que $|\mathcal{L}| = |\mathbb{N}_2|$
Por tanto, $|\mathcal{M}| < |\mathcal{L}|$ Es decir hay más lenguajes que MT. Así, debe haber lenguajes no reconocibles por MT alguna.
$\varphi : \mathcal{L} \to \mathbb{N}_2$ Sea $S_0, S_1, S_2, ...$ el orden canónico de $Sigma^*$, y sea $\mathcal{L}$ un lenguaje sobre $Sigma^*$ ■

¿En general $S \sim P(S)$? <u>*No*</u>
Sea $f : S \to P(S)$. Veamos que f no es suprayectiva. Proponemos $T = \{s \in S | s \notin f(s)\}$ Si $s \in T$ entonces $s \notin f(s)$ $(T \cup (S - T) = S)$ Si $s \notin T$ entonces $s \in f(s)$ Por tanto $T \neq f(s)$ $T \notin Im(f)$■

A, E, EQ $A_{TM} = \{< M\omega > |$M es una MT que acepta a la cadena$\omega\}$
Obs. A_{TM} es reconocible. U = Con entrada $< M, w >$ donde M es una MT

1. Simula a M con entrada w
2. Si M acepta, <u>*acepta*</u>, si M rechaza, <u>*rechaza*</u>

Prop. A_{TM} no es decidible Dem. Supongamos que sí, y sea R un decididor de A_{TM} $R(< M, w >) = \{$<u>*acepta*</u>si M acepta a w, <u>*rechaza*</u>si M rechaza a w$\}$ Construimos una MT D que usa a R como subrutina D = "Con entrada $< M >$ dada M es una MT:



1. Ejecuta R con entrada $< M, < M >>$

2. Si R acepta, *rechaza*; si R rechaza, *acepta*"

$D(< M >) = \{$*acepta*si M no acepta a $< M >$; *rechaza*si M acepta a $< M >\}$
$D(< D >) = \{$*acepta*si D rechaza a $< D >$; *rechaza*si D acepta a $< D >\}$! D *acepta* a D si y sólo si D *rechaza* a D! ■

Def.: Un lenguaje L sobre Σ es $co - reconocible$ si $\Sigma(\Sigma^* - L)$ es reconocible.
Prop. Un lenguaje es decidible si y sólo si es reconocible y co-reconocible.
Dem: Sea A un lenguaje. Si A es decidible, entonces es reconocible. Sea M un decididor para A. M = "Con entrada w:

1. Ejecuta M con entrada w

2. Si M acepta, *rechaza* Si M rechaza, *acepta*

L(M')= A Si A es reconocible y co-reconocible. Proponemos una MT M que decide a A que usa a M_1 y M_2 que reconocen a A y $\overline{\wedge}A$ respectivamente. M = "Con entrada w:

1. Simula en paralelo a M_1 y M_2 con entrada w

2. Si M_1 acepta, *acepta*

3. Si M_2 acepta, *rechaza*■

Cor: El complemento de A_{TM} no es reconocible.
Dem: A_{TM} es reconocible. Si su complemento fuera reconocible, entonces A_{TM} sería decidible pero no lo es ■

$H_{TM} = \{< M, w > |$M es una MT y M se detiene con entrada w$\}$
Prop: H_{TM} es indecible
Dem: Supongamos que R decide H_{TM} Proponemos la MT S, dada por: S = " Con entrada $< M, w >$, donde M es una MT:

1. Ejecuta a R con entrada ¡M,w¿

2. Si R rechaza, *rechaza*

3. Simula a M con entrada w

4. Si M acepta, *acepta*, si no, *rechaza*"

Claramente, S decide a A_{TM}! ■
$E_{TM} = \{< M > |L(M) = \emptyset\}$
Prop E_{TM} no es decidible.
Dem. Por contradicción. Supongamos que sí es decidible y sea R un decididor para E_{TM}. Construimos a S un decididor para A_{TM} dado por: S : "Con entrada $< M, w >$ donde M es una MT:

1. Construye a la MT M_1 dada por: M_1 = "Con entrada x:

   (a) Si $x \neq w$, rechaza

   (b) Si $x = w$, corre a M con entrada w y acepta si M lo hace".

2. Ejecuta R con entrada $< M_1 >$

3. Si R acepta, *rechaza*. Si R rechaza, *acepta*".

El lenguaje de M_1 es $|w|$ o $\emptyset$ dependiendo de si M acepta o no a w respectivamente. Por tanto, S es un decididor para A_{TM}■
Entonces E_{TM} no es decidible
$REG_{TM} = \{< M > |M$es una Mt y L(M) es regular$\}$
Prop. REG_{TM} es indecidible
Dem: Por contradicción. Sea R un decididor para REG_{TM} Proponemos la MT S dada por: S = "
Con entrada $< M, w >$, donde M es una MT:

1. Construye a la MT M_2 dada por: M_2 = " Con entrada x:

   (a) Si x es de la forma 0^2 acepta

   (b) Si no, ejecuta M con entrada w y *acepta* a x si M acepta a w".

2. Ejecuta R con entrada $< M_2 >$

3. Si R acepta, *acepta*. Si R rechaza, *rechaza*.

Como entonces S decide a A_{TM}■

2.1 Reducibilidad

Tenemos dos problemas: Problema 1 (Entrada: I_1 Q: Q_1) Problema 2(Entrada: I_2 Q: Q_2)
Si tenemos un algoritmo M_1 para resolver Prob 2, Y un algoritmo M_2 que tiene una instancia I_1 del Prob 1 y la transforma en una instancia I_2 del Prob 2, de tal forma que I_2 es una sí-instancia del Prob. 2 si y sólo si I_1 es una sí-instancia del Prob 1, entonces
Proponemos M_3 que resuelve Prob 1.
$I_1 \rightarrow^{M_2} I_2$ Si M_1 responde sí, tenemos que I_2 es una sí-instancia $\rightarrow I_1$ es una sí-instancia Si M_1 responde no, tenemos que I_2 es una no-instancia $\rightarrow I_1$ es una no-instancia
Si Prob. 1 se reduce a Prob. 2 y hay un algoritmo para resolver Prob. 2, entonces hay un algoritmo para resolver el problema 1.
Si sabemos que no hay un algoritmo para resolver Prob. 1 y podemos reducir Prob. 1 a Prob. 2, entonces no hay un algoritmo para resolver al Prob. 2
Problema 1 E_{TM} Entrada: $< M >$ con M una MT Pregunta: ¿L(M) = $\emptyset$?
Problema 2 $EQ_{TM} = \{< M_1, M_2 > |M_1$ y M_2 son MT y $L(M_1) = L(M_2)\}$ Entrada: $< M_1, M_2 >$ con M_1 y M_2 MT Pregunta: ¿L(M_1) = L(M_2)?
Vamos a mostrar que el problema 1 se reduce al problema 2 $< M > \rightarrow^A < M, M_2 >$ donde M_2 es una MT que rechaza a todas sus entradas. Si L(M) = $\emptyset$ entonces $L(M) = L(M_2)$ Si $L(M) \neq \emptyset$ entonces $L(M) \neq L(M_2)$
Prop. EQ_{TM} es indecidible.
Dem: Por contradicción. Suponemos que R es una MT que decide a EQ_{TM}. Proponemos una MT S. S = "Con entrada $< M >$ donde M es una MT:

1. Ejecuta R con entrada $< M, M_2 >$ donde M_2 es una MT que rechaza todas las entradas

2. Si R acepta, *acepta*. Si R rechaza, *rechaza*. Entonces si R acepta a $< M, M_2 >$ que dado que $L(M_2) = \emptyset$, entonces $L(M) = \emptyset$. Si R rechaza a $< M, M_2 >$ entonces $L(M) \neq \emptyset$ luego, S acepta a $< M >$ si y sólo si $L(M) \neq \emptyset$. Por lo tanto S decide a E_{TM}!

Def. Una función $f : \Sigma^* \to \Sigma^*$ es *computable* si existe una MT M que con cualquier entrada w se detiene con una reducción $f(w)$ en su cinta
Def. Un lenguaje A es reducible si por funciones (muchas o una) al len. B, $A \leq_M B$ si existe una función computable $f_:\Sigma^* \to \Sigma^*$ tal que para cada $w \in \Sigma^*$, $w \in A$ si y solo si $f(w) \in B$
Teo. Si $A \leq_M B$ y B es decidible, entonces A es decidible.
Dem: Sea M un decididor para B y sea f la reducción de A a B. Proponemos a N. N = "Con entrada w:

1. Calcula $f(w)$
2. Ejecuta M con entrada $f(w)$
3. Responde lo mismo que M ".

Si $w \in A$, por la def. de reducción, $f(w) \in B$, y por tanto M acepta a $f(w)$ y N acepta a w. Si $w \notin A$, $f(w) \notin B$, y M rechaza a $f(w)$ y N rechaza. Luego, N decide a A ■
Cor Si $A \leq_M B$ y A no es decidible, entonces B no es decidible.
PropH_{TM} es indecidible.
Dem: Veamos que el problema del paro se reduce al problema de la aceptación $A_{TM} \leq H_{TM}$ Proponemos a F tal que F = "Con entrada $< M, w >$ donde M es una MT:

1. Construye la MT M' dado que: M' = "Con entrada x:
   (a) Corre a M con entrada x
   (b) Si M acepta, *acepta*
   (c) Si M rechaza, *se cicla*
2. Devuelve $< M', w >$"

Si $< M, w > \in A_{TM}$, entonces $< M', w > \in H_{TM}$.
Si $< M, w > \notin A_{TM}$, entonces $< M', w > \notin H_{TM}$■
$A \leq_M B$ B decidible entonces A decidible A indecidible entonces B indecidible
$A_{TM} \leq H_{TM}$
$E_{TM} \leq EQ_{TM}$ $< M > \to < M, M_2 >$, M_2 rechaza todas las codificaciones
$< M > \in E_{TM} \leftrightarrow L(M) = \emptyset \leftrightarrow L(M) = L(M_2) \leftrightarrow < M_1, M_2 > \in EQ_{TM}$
E_{TM} es indecidible $\overline{E_{TM}}$ es indecidible $< M, w >$ M_1 = "Con entrada x

1. Si $x \neq w$, *rechaza*
2. Ejecuta M con entrada w
3. Si M acepta, *acepta*

$L(M_1) = \emptyset$si w $\notin L(M)|w|$si w $\in L(M)$
$< M, w > \to < M_1 >$
$< M_1, w > \in A_{TM} \leftrightarrow$ M acepta a w $\leftrightarrow L(M_1) \neq \emptyset \leftrightarrow < M_1 > \notin E_{TM}$
Obs L es decidible si y sólo si $\overline{L}$ complemento es decidible.
Dem: Si R decide a L proponemos a S dada por: S = "Con entrada w

1. Ejecuta R con entrada w
2. Si R acepta, *rechaza* y s R rechaza, *acepta*

Obs Si $A \leq_M B$ entonces $\overline{A} \leq_M \overline{B}$. Dem: f computable tal que $w \in A$ si y sólo si $f(w) \in B$ luego $e \notin A$ si y sólo si $f(w) \notin B$
Obs Si $A \leq_M B$ y B es reconocible, entonces A es reconocible R es un reconocedor $f(w) \in B$ si y sólo si $w \in A$ S = "Con entrada w

1. Calcular f(w)
2. Ejecutar R con entrada f(w)
3. Responder lo mismo que R"

$S(w) = R(f(w))$
$A_{TM} \leq \overline{E_{TM}} \Rightarrow \overline{A_{TM}} \leq E_{TM}$
$\overline{E_{TM}}$ es reconocible
$A_{TM} \leq_M \overline{E_{TM}}$ ¿$A_{TM} \leq_M E_{TM}$? No
$\overline{A_{TM}} \leq_M \overline{E_{TM}} \Rightarrow \overline{E_{TM}}$ no es reconocible !

Prop EQ_{TM} no es reconocible ni co-reconocible.
$A_{TM} \leq \overline{EQ_{TM}}$ $A_{TM} \leq EQ_{TM}$ Dem: F = "Con entrada ¡M,w¿ donde M es una MT

1. Construimos las MT M_1 y M_2 dadas por M_1 = "Con entrada x:
   (a) *Rechaza*"

   M_2 = "Con entrada x:
   (a) Ejecuta a M con entrada w. Si M acepta, *acepta*
2. Devuelve $< M_1, M_2 >$

Notemos que $L(M_1) = \emptyset$ y $L(M_2) = \Sigma^*$ si M acepta a w o $\emptyset$ si M no acepta a w.
Si $< M, w > \in A_{TM}$, entonces $L(M_2) = \Sigma^*$ y $< M_1, M_2 > \in \overline{EQ_{TM}}$.
Si $< M_1, M_2 > \in \overline{EQ_{TM}}$, entonces $L(M_2) \neq L(M_1)$, entonces $L(M_2) = \Sigma^*$ y por tanto M acepta a w y $< M, w > \in A_{TM}$
Luego F calcula una reducción $A_{TM} \leq_M \overline{EQ_{TM}}$
Si hacemos que M_1 acepte en lugar de rechazar, tenemos que: Si $< M, w > \in A_{TM}$, entonces $L(M_2) = \Sigma^*$ y $< M_1, M_2 > \in EQ_{TM}$.
Si $< M_1, M_2 > \in EQ_{TM}$, entonces $L(M_2) \neq L(M_1)$, entonces $L(M_2) = \Sigma^*$ y por tanto M acepta a w y $< M, w > \in A_{TM}$

3 Complejidad

Vamos a trabajar únicamente con máquinas de Turing decidibles
Sea M una MT que se detiene en todas sus entradas. La complejidad (en tiempo) de M es la función $f : \mathbb{N} \to \mathbb{N}$ tal que $f(n)$ es el mejor tiempo que puede tardarse M en ejecutarse con una entrada de longitud n. Si $f(n)$ es el tiempo de ejecución de M, diremos que M corre en tiempo $f(n)$ o que M es de tiempo $f(n)$.
Sea $t : \mathbb{N} \to \mathbb{R}$ una función. La clase de de complejidad en tiempo TIME(t(n)) es el conjunto de todos los lenguajes decidibles en tiempo $\mathcal{O}(t(n))$ por una MT
$L = \{0^n 1^n\} n \in \mathbb{N}$ M = "Con entrada w:

1. Recorre la cinta y si encuentra un 1 a la derecha de un 0, *rechaza* ($\mathcal{O}(n)$)
2. Repite mientras haya 0 y 1 en la cinta

3. Recorre la cinta(entrada) y borra el primer 0 y el primer 1. ($\mathcal{O}(n) \cdot \frac{n}{2}$)
4. Si sobra algún 0 o algún 1 en la cinta *rechaza*. Si no, *acepta*". ($\mathcal{O}(n)$)

$M \in TIME(n^2)$ M es de tiempo $\mathcal{O}(n^2)$ $L \in TIME(t(n))$
$TIME(n) \subseteq TIME(n^2)$
M_2 (Máquina de Turing con 2 cintas)= "Con entrada w:

1. Recorre la entrada y *rechaza* si hay un 0 a la derecha de 1. ($\mathcal{O}(n)$)
2. Repite mientras haya 0 y en 1 en la cinta ($\mathcal{O}(\log n)$)
3. Revisa si el número de 0 y 1 es par o impar. Si es impar, *rechaza* ($\mathcal{O}(n)$)
4. Recorre la cinta borrando un 0 si y uno no y un 1 si y uno no, empezando por el primero. ($\mathcal{O}(n)$)
5. Si hay 0 o 1 en la cinta, *rechaza*. Si no, *acepta*" ($\mathcal{O}(n)$)

$\mathcal{O}(n) + \mathcal{O}(\log n) \cdot 2 \cdot \mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n \log n)$
M_2 corre en tiempo $\mathcal{O}(n \log n)$ ó $\Theta(n \log n)$
$L \in TIME(n \log n)$ $L \in TIME(n^2)$
M_3 (Máquina de Turing con 3 cintas)= "Con entrada w:

1. Recorre la entrada y copia los 0 en la cinta 2 y los 1 en la cinta 3 ($\mathcal{O}(n)$)
2. Repite mientras haya 0 en la cinta 2 y 1 en la cinta 3 ($\mathcal{O}(n)$)
3. Borra un 0 en la cinta 2 y un 1 en la cinta 3. ($\mathcal{O}(1)$)
4. Si los 0 se acaban antes que los 1, o viceversa, *rechaza*. De otro modo *acepta*". ($\mathcal{O}(n)$)

M_3 corre en tiempo $\mathcal{O}(n)$
$L \in TIME(n \log n)$ Cambiando el modelo $\rightarrow L \in TIME(n)$

Teo Sea t(n) una función con $t(n) \geq n$ Entonces toda MT multicinta de tiempo t(n) tiene una MT de una sola cinta equivalente y de tiempo $\mathcal{O}(t(n^2))$
Dem: Utilizamos la simulación de una MT multicinta M con una MT de una sola cinta S. Para simular un paso de M, S recorre la entrada y ubica el símbolo que lee cada cabezal virtual; esto lleva tiempo la longitud de la cadena actualmente en su cinta. Esta longitud es la suma de las longitudes de los contenidos de todas las cintas de M. Cada cinta de M contiene a lo más t(n) símbolos (pues se ejecuta por a lo más tiempo t(n)) En total, este paso se lleva tiempo $k \cdot t(n) \in \mathcal{O}(t(n))$ Después, recorre nuevamente la cinta, actualizando las acciones de M. Quizá, haciendo k shifts en el proceso. Este recorrido junto con los shifts se lleva tiempo $k \cdot \mathcal{O}(t(n))$ que es $\mathcal{O}(t(n))$. Así, S se tarda $\mathcal{O}(t(n))$ en simular un paso de M, y tiene que simular t(n) pasos, luego el tiempo total es $\mathcal{O}(t(n)^2)$■.
N es una MTN t(n) $\rightarrow$ existe M MTD equivalente $\mathcal{O}(2^{t(n)})$
Toda MT multicinta M_1 tiene una MT M_2 de una cinta tal que si M_1 corre en un tiempo t(n), entonces M_2 es de tiempo $\mathcal{O}(t(n^2))$

Def Sea t: $\mathbb{N} \rightarrow \mathbb{N}$ tal que $t(n) \geq n$ para cada $n \in \mathbb{N}$. Decimos que la MTN N corre un tiempo t(n) si t(n) es el máximo número de pasos que debe realizar N para detenerse con una entrada de longitud n. N es un decididor (Todas las ramas terminan).
M simula a N

1. cinta w - entrada de N

2. cinta de trabajo

3. generar dirección

Teo Si N es una MTN de tiempo t(n) que es un decididor, donde $t(n) \geq n$ p.c. $n \in \mathbb{N}$, entonces existe una MTD, D, tal que D es equivalente a N y es de tiempo $2^{\mathcal{O}(t(n))}$.
Dem: Consideremos la simulación de N por una MTD de 3 cintas vista anteriormente. Con una entrada de longitud n, cada rama de la ejecución no determinista de N tiene longitud a lo más t(n).Además cada nodo en el árbol tiene a lo más b hijos. Hay a lo más $b^{t(n)}$ hojas en el árbol y a lo más, el mismo número de vértices internos , i.e. hay $\mathcal{O}(b^{t(n)})$ vértices en el árbol de ejecución de N. El tiempo que toma a M iniciar en la raíz y simular una rama de la ejecución de N hasta un vértice dado, es a lo más t(n). Luego, hay $\mathcal{O}(b^{t(n)})$ vértices en el árbol y simular a M para llegar a cada uno de ellos en tiempo t(n), i.e., la simulación completa toma tiempo $t(n) \cdot \mathcal{O}(b^{t(n)}) = \mathcal{O}(b^{t(n)}) = \mathcal{O}(2^{t(n)}) = 2^{\mathcal{O}(t(n))}$. Como M tiene 3 cintas, la podemos simular con una MT de una sola cinta en tiempo $2^{2\mathcal{O}(t(n))} = 2^{\mathcal{O}(2t(n))} = 2^{\mathcal{O}(t(n))}$■

Def La clase P es la clase de lenguajes decidible por una MTD de una sola cinta en tiempo polinomial, i.e. $P = \bigcup_{t \in \mathbb{N}} TIME(n^t)$
Def La clase NP es la clase de lenguajes decidible por una MTN de una sola cinta en tiempo polinomial, i.e. $NP = \bigcup_{t \in \mathbb{N}} NTIME(n^t)$
$NTIME(t(n))$ = Todos los lenguajes L tales que hay una MTN de tiempo t(n) que decide a L.
NP es Non-deterministic Polynomial, i.e. una MT no determinista lo puede decidir en tiempo Polinomial.
PATH$\{< D, s, t >$— hay una st-trayectoria (camino que no repite vértices) dirigida en D$\}$
Prop: PATH $\in$ P
Dem: Proponemos a la MT M dada por M = "Con entrada ¡D, s, t¿ donde D es una digráfica y s y t son vértices de D:

1. Marca S.

2. Repite hasta que no marquen vértices nuevos.

3. Revisa cada flecha de D. Si hay alguna cuya cola esté marcada y cuya cabeza no esté marcada, marca la cabeza.

4. Si t está marcado, *acepta*. Si no, *rechaza*

Los pasos 1 y 4 se ejecutan una única vez y toman tiempo $\mathcal{O}(n)$. A lo más, el paso 3 se repite $|V|(es\mathcal{O}(n))$ veces. Cada ejecución de 3 es $\mathcal{O}(n)$. Luego, todas las ejecuciones de 3 se pueden hacer en $\mathcal{O}(n^2)$.
Así, M es de tiempo $\mathcal{O}(n^2)$■
RELPRIME = $\{< x, y > | x$ y y son primos relativos$\}$
Prop : RELPRIME $\in$ P
Dem: Proponemos a E una MT dada por
E = "Con entrada $< x, y >$ donde x y y son enteros positivos

1. Repite hasta que $y = 0$

2. Asigna $x \leftarrow x(mody)$

3. Intercambia x y y

4. Devuelve x

Proponemos a M dada por:
M = "Con entrada $< x, y >$ donde x y y son enteros positivos

1. Ejecuta E con entrada $< x, y >$
2. Si E devuelve 1 *acepta*. Si no, *rechaza*

Como cada paso individual se puede hacer en tiempo polinomial, basta ver cuantas veces se repiten los pasos 2 y 3.
Si $\frac{x}{2} \geq y$, $x(\mod y) < y \leq \frac{x}{2}$ Si $\frac{x}{2} \leq y$, $x(\mod y) = x - y < \frac{x}{2}$
Cada dos vueltas del ciclo, el valor de x y el de y disminuyen a la mitad. Luego, estos pasos se realizan $\mathcal{O}(\log(n))$ veces■
EQ_{LLC} es indecidible.

Sí-certificado. Algoritmo para verificar el certificado (simple).
Un *verificador* para el lenguaje A es una MT decididora V tal que $A = \{w|$V acepta $< w, c >$ p.c cadena c$\}$ La complejidad de V se mide en términos de la longitud de w, luego, un verificador polinomial para A es un verificador que corre en tiempo polinomial respecto a $|w|$.

Ejemplo: L = $\{< G > |G$es k-coloreable$\}$ V = "Con entrada $< G, c >$ donde G es una gráfica y c es una k-coloración de G:

1. Revisa que cada vértice aparezca una única vez en c, si no, *rechaza* $\mathcal{O}(|V|^2)$
2. Para cada arista e de G, verifica si los extremos de e tienen colores distintos. Si no, *rechaza* $\mathcal{O}(|E|)$
3. Revisa que haya a lo más k-colores en c. Si no, *rechaza*. $\mathcal{O}(|V|)$
4. *Acepta*

El certificado es una cadena, verificador es un algoritmo.
Ejemplos de problemas con sus verificadores:
k-col <G, coloración>
ciclo hamiltoniano <G, ciclo hamiltoniano>
graf conexa <G, árbol generador>
sat < ϕ,τ(asignación de verdad)>

Teo Un lenguaje L está en NP si y sólo si tiene un verificador polinomial.
Def Un verificador para un lenguaje A es una MT decididora tal que $A = \{w|V$acepta $< w, c >$ p.a. cadena c$\}$
HAMPATH = $\{< D, s, t > |D$es una digráfica, s y t vértices de D y tales que existe una s-t trayectoria hamiltoniana$\}$
Prop. HAMPATH $\in NP$
Dem: Proponemos una MTN N tal que:
N = "Con entrada $< D, s, t >$ donde D es una digráfica, s,t$\in V_D$:

1. Escribe una lista de enteros $p_1, .., p_{|V|}$ cada número en la lista se elige de manera no determinista entre 1 y —V— $\mathcal{O}(|V|)$
2. Si hay repeticiones en la lista *rechaza* $\mathcal{O}(|V|^2)$
3. Revisa si $p_1 = s$ y $p_{|V|} = t$, si no, *rechaza* $\mathcal{O}(|V|)$
4. Para cada i entre 1 y —V—-1, revisa que $(p_i, p_i + 1)$ sea una flecha en D. Si no *rechaza* $\mathcal{O}(|V|^t)$ $t \in \mathbb{N}$

5. Acepta

N corre en tiempo polinomial respecto a —V—, por lo tanto HAMPATH ∈ NP ■
Teorema Un lenguaje L está en NP si y sólo si tiene un verificador polinomial
Dem: Si V es un verificador polinomial para L, proponemos la MTN N que decide a L dada por: Supongamos que V es de tiempo n^t N: "Con entrada w de longitud n:

1. Elige de manera no determinista una cadena c de longitud a lo más n^t
2. Ejecuta V con entrada $< w, c >$
3. Si V acepta, *acepta*. Si no, *rechaza* $\mathcal{O}(n^t)$

Si L es decidido por la MTN N y la rama de la ejecución no determinista está codificada por C, entonces proponemos a V dada por: V="Con entrada $< w, c >$

1. Simula a N con entrada w usando cada símbolo de c como la siguiente elección en la ejecución no determinista de N que acepta a w. $\mathcal{O}$ = El tiempo que tarda N en decidir a w
2. Si esta rama de la ejecución de N acepta, *acepta*. Si no, *rechaza*■ $\mathcal{O}(1)$

$L \in NP$ L = $\{w|$V acepta $< w, c >$ para alguna cadena c$\}$ V verificador polinomial para L:

1. "Adivinar" una solución
2. Verificar que la solución es correcta

Si G es una gráfica, un clan de G es un subconjunto C de V tal que G[c](gráfica inducida) es completa. Un k-clan es un clan de cardinalidad k. El máximo entero k tal que G tiene un k-clan es el número de clan de G ω_G
CLAN= $\{< G, k > |$G es una gráfica con un k-clan$\}$
(Mapping reducibility) Existe una MT que computa f:$\Sigma^* \to \Sigma^*$ en tiempo polinomial, $\mathcal{O}(n^k)$ tal que $A \leq_m B$, $w \in A$ si y sólo si $f(w) \in B$. M decide B en tiempo polinomial. M se ejecuta con entrada f(w), en —f(w)— en $\mathcal{O}(n^k)$, luego su tiempo de ejecución es $\mathcal{O}(|f(w)|^l) = \mathcal{O}((n^k)^l) = \mathcal{O}((n^{k \cdot l})$ F con entrada w computa f(w) en tiempo $\mathcal{O}(|w|^k) = \mathcal{O}(n^k)$
Def. Sean A y B lenguajes. Decimos que A es reducible en tiempo polinomial mediante funciones a B, $A \leq_p B$ si existe una función computable en tiempo polinomial $f : \Sigma^* \to \Sigma^*$ tal que, para cada $w \in \Sigma^*$ se tiene: $w \in A$ si y sólo si $f(w) \in B$. Se dice que f es una reducción polinomial de A en B.
Prop. Si $A \leq_P B$ y $B \in P$, entonces $A \in P$ Dem. Sea M la MT que decide a B en tiempo polinomial, y sea f la reducción polinomial de A en B. Proponemos a S una MT que decide a A, dada por: S: "Con entrada w:

1. Calcula f(w)
2. Ejecuta M con entrada f(w) y responde lo mismo que M".

Para ver que S decide a A, consideremos $w \in A$. Entonces $f(w) \in B$ y por lo tanto M acepta a f(w). Luego, S acepta a w. Recíprocamente si S acepta a w, entonces M acepta a f(w), i.e. $f(w) \in B$ y como f es una reducción, $w \in A$. Se sigue que S decide a A. Claramente el paso 1 se realiza en tiempo polinomial or la definición de reducción polinomial y el paso 2 también, pues la composición de polinomios es un polinomio. (Si f(w) es $\mathcal{O}(n^k)$ M con entrada f(w) tarda $\mathcal{O}(n^k)^l = \mathcal{O}(n^{k \cdot l})$)
Si $A \leq_P B$ y $A \notin P$ entonces $B \notin P$
SAT = $\{< \varphi > |\varphi$ — φ es una fórmula booleana satisfacible$\}$
Una literal es una variable proporcional o su negación.
Una clausula es una disyunción con literales.
Una fórmula booleana está en FNC si es una conjunción de disyunciones.

$SAT_{FNC} = \{< \varphi > | \varphi$ es una función booleana en FNC satisfacible $\}$
$SAT_{FNC} \leq_P SAT$
Una fórmula está en 3-FNC si es una conjunción de clausulas de tamaño 3.
3SAT = $\{< \varphi > | \varphi$ es una fórmula booleana satisfacible en 3-FNC $\}$
CLAN = $\{< G, k > |$ G es una gráfica con un clan de cardinalidad k $\}$
Prop. 3SAT $\leq_P$ CLAN Modela el problema de la izquierda con el de la derecha. Sea $\varphi \in 3FNC$, entonces $\varphi = \cap_{i=1}^{k}(a_i \cup b_i \cup c)$ $(f(< \varphi >) =< G, k >)$ Para cada clausula en φ creamos 3 vértices en G, etiquetados con las literales correspondientes en la clausula $(V = \cup_{i=1}^{k}\{a_i, b_i, c_i\})$ Cualesquiera dos vértices de G son adyacentes, excepto:

1. Vértices correspondientes a la misma tripleta (De la misma clausula)
2. Vértices con etiquetas que generan una contradicción, e.g. x_i y $\overline{x_i}$

Veamos que $< \varphi > \in 3SAT$ si y sólo si $< G, k > \in CLAN$.
Si $< \varphi > \in 3SAT$ existe τ asignación de verdad que satisface a φ. Sea $\{x_i\}_{\{i \in 1 \ldots k\}}$ un conjunto de literañes de φ tal que $\tau(x_i) = 1$ y x_i pertenece a la i-ésima cláusula. Notemos que al recibir todas las x_i el valor 1 bajo τ, tenemos que no es posible encontrar a una literal y a su negación en τ. Se sigue que en G los vértices correspondientes a $\{x_i\}_{i=1}^{k}$ es un clan de tamaño k. Luego $< G, k > \in CLAN$
Si $< G, k > \in CLAN$, entonces G tiene un clan de tamaño k, $S \subseteq V$. Por la construcción de G no hay dos vértices de S en la misma tripleta, y no hay dos vértices de S tales que sus etiquetas sean contradictorias, por lo tanto podemos asignar 1 a cada terminal correspondiente a un vértice en S. Si llamamos τ a alguna asignación obtenida al extender la asignación antes descrita a todas las literales de φ, entonces τ satisface a φ (pues le asigna el valor 1 al menos una variable en cada clausula).
Así, f es una reducción $|V| = 3k$ $|E| \cdot \mathcal{O}(k^2)$
STABLE = $\{< G, E > |$G es una gráfica con un conjunto independiente de tamaño k$\}$
CLIQUE = Todos son adyacentes dos a dos
$CLIQUE \leq_P STABLE < G, k > \mapsto < H, l >$ $H = \overline{G}$ $l = k$
$< G, k > \in CLIQUE$, entonces G tiene un clan de tamaño k, digamos S. En $\overline{G}$ S es un conjunto independiente de tamaño k. Si $< \overline{G}, k > \in STABLE$ entonces $\overline{G}$ tiene un conjunto independiente de tamaño k y pi tanto $\overline{G} = G$ tiene un clan de tamaño k, entonces $< G, k > \in CLIQUE$
Def Sea G una digráfica. Un subconjunto $S \in V$ es <u>convexo</u> si todos los vértices de cualquier geodésica (trayectoria de longitud mínima) entre pares de vértices distintos están contenidos en S. Usualmente, por conjunto convexo nos referimos a uno distinto de V.
CONVEX = $\{< G, k > |$G tiene un conjunto convexo de tamaño k$\}$
$CLIQUE \leq CONVEX < G, k > \mapsto < H, l >$
H se obtiene de G al añadir dos vértices nuevos, x y y, y hacerlos adyacentes a cada vértice en G, pero no entre sí.
Si G tiene un k-clan S, $S \cup \{x\}$ es un clan, y por tanto es un conjunto convexo. Por tanto, H tiene un (k+1)-conjunto convexo
$S \subseteq V_H$ es conexa.
$S \subseteq V_G$
Afirmamos S es un clan. De otro modo, $x \in S$ Si $x \in S$, entonces $y \notin S$. Nuevamente S es un clan, pues de otro modo $y \in S$
Por lo tanto G tiene un k-clan■
$|V_H| = |V_G| + 2$ $|E_H| = |E_G| + 2|V_G|$
CLIQUE $\leq_P$ CONVEX
DConvex = $\{< D, k > |D$es una digráfica con un conjunto convexo de tamaño k$\}$
Prop. CLIQUE $\leq$ DConvex
La gráfica será:

- Bipartita
- Sin flechas simétricas
- Con cuello 6 (ciclo más corto será de longitud 6)

Dem:
Gadgets Por cada vértice de la gráfica hay un hexágono y para cada arista de la gráfica se unirá el vértice x del primer hexágono al y del segundo hexagono y el x del segundo hexagono al y del primer hexágono.
La estructura de control será una trayectoria dirigida
$|V_G| = 6|V_G| + 4$ $|A_G| = 8|V_G| + 2|E_G| + 3$
Obs.

- Si $n \in V_G$ y C un convexo de D $|C| \geq 2$, tal que $C \cap V_{Hu} \neq \emptyset$, entonces $V_{Hu} \in C$
- Sea C un convexo de D_1 $|C| \geq 2$. Si $z_i \in C$, p.a. $i \in \{1...4\}$ entonces $C = V_0$
- Sean $u, r \in V_G$ t.q. $d(u, r) \geq 2$, si C es un convexo en D t.q. $V_{Hu} \cap C \neq \emptyset \neq V_{Hu} \cap C$, ent. $C = V_D$
- Si S es un clan de G, ent. $C = \cup_{u \in S} Hu$ es un convexo en D. Si G tiene un k-clan, entonces D tiene un 6k conjunto convexo.

Un lenguaje A es NP-completo si:

- $A \in NP$
- Para cada $B \in NP$, $B \leq_P A$

Si hubiera un algoritmo polinomial para resolver A, entonces tenemos un algoritmo polinomial para resolver cualquier problema en NP.
Si L es NP-c y tenemos al lenguaje A t.q.

- $A \in NP$
- $L \leq_P A$, entonces A es NP-c

$4 - col = \{< G > | G \text{es una gráfica 4-coloreable}\}$
Prop 4-col $\leq_P$ SAT
$G \mapsto \varphi$
φ es satisfacible si y solo si G es 4-coloreable
x_{vc} El vértice v recibe el color c
$C = \{a, b, c, d\}$
$\varphi_{color} = \wedge_{v \in V_G}(x_{va} \cap \overline{x_{vb}} \cap \overline{x_{vc}} \cap \overline{x_{vd}}) \cup (\overline{x_{va}} \cap x_{vb} \cap \overline{x_{vc}} \cap \overline{x_{vd}}) \cup (\overline{x_{va}} \cap \overline{x_{vb}} \cap x_{vc} \cap \overline{x_{vd}}) \cup (\overline{x_{va}} \cap \overline{x_{vb}} \cap \overline{x_{vc}} \cap x_{vd})$
$\varphi_{propia} = \wedge_{uv \in E_G}(\neg(x_{ua} \cap x_{\sigma a}) \cap \neg(x_{ub} \cap x_{vb}) \cap \neg(x_{uc} \cap x_{vc}) \cap \neg(x_{ud} \cap x_{vd}))$
$\varphi = \varphi_{color} \cap \varphi_{propia}$
Si G es 4-col, entonces φ satisfacible. φ satisfacible, entonces G es 4-col.
k-coloracion se puede reducir a SAT en tiempo lineal
A es NP-completo
$SubsetSum \in NP$
$SubsetSum \leq_P A$
M decide a A en tiempo polinomial
Teo Cook-Levin SAT es NP-completo
Dem: Sea $L \in NP$, entonces existe N una MTN que decide a L, el tiempo de ejecución de N es $\mathcal{O}(n^k)$ p.a. $k \in \mathbb{Z}^+$

$C = Q \cup \Gamma \cup \{\#\}$
$x_{i,j,s}$ se interpreta como $T_{i,j} = s$
$\varphi_{cell} = \wedge_{1 \leq i,j \leq n^k}[(\vee_{s \in C} x_{i,j,s}) \& (\wedge_{s,t \in C \ s \neq t}(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}))]$ $\mathcal{O}(n^{2k}) \cdot \mathcal{O}(l) \cdot \mathcal{O}(l^2)$
$\varphi_{start} = x_{1,1,\#} \& x_{1,2,q_0} \& ... \&$
$\varphi_{accept} = \vee_{1 \leq i,j \leq n^k} x_{i,j,q_a}$
$\delta(q_1, a) = \{(q_1, b, R)\}$ $\delta(q_1, b) = \{(q_1, c, R), (q_1, a, R)\}$
$\varphi_{start} = x_{1,1,\#} \& x_{2,2,a} \& x_{2,3,q_2} \& x_{2,4,q_2} ... \&$
$\underline{Afirmacion}$: Si el primer renglón es la configuración inicial y cada ventana es legal, entonces cada renglón de la tabla es una configuración que se sigue de la anterior.
Dem: Si en la ventana no aparecen estados entonces, para que sea legal, el símbolo central es igual arriba y abajo. Si en la posición central superior aparece un estado, las tres posiciones de abajo corresponden a una transición correcta gracias a que la ventana es legal ■
$\varphi_{move} = \vee_{1 \leq i < n^k, 1 < j < n^k}()$
Ventana: Subarreglo de 2x3, determinado por su posición central superior
$\varphi = \varphi_{cell} \& \varphi_{start} \& \varphi_{accept} \& \varphi_{move}$
φ es satisfacible si y sólo si existe una ejecución de N que acepta a w
$\varphi_{accept} = \vee x \mathcal{O}(n^{2k}) \cdot \mathcal{O}(l^6) = \mathcal{O}(n^{2k})$
$|\varphi| \in \mathcal{O}(n^{2k})$

3.1 Espacio

Sea M una MTD que se detiene con todas sus entradas, la complejidad en espacio de M es la función $f : \mathbb{N} \to \mathbb{N}$ donde $f(n)$ es el máximo número de celdas que lee M en una ejecución con una entrada de tamaño n. Si la complejidad en tiempo de M es f decimos que corre en espacio $\mathcal{O}(f(n))$. Se define análogamente la complejidad en espacio para una MTN en la que todas las ramas de su ejecución se detienen con todas las entradas.
Proposicion: SAT se puede decidir en espacio polinomial
Dem: Sea M una MT tal que M = "Con entrada $< \varphi >$ donde φ es una fórmula booleana.

1. Por cada asignación de verdad de las variables $x_1, ..., x_m$ de φ
2. Evalúa φ en la asignación de verdad
3. Si en algún momento φ se evalúa a 1, acepta. Si no, rechaza. "

$\mathcal{O}(m)$ genera todas las asignaciones de verdad. $\mathcal{O}(m)$ es $\mathcal{O}(|\varphi|)$
$\underline{ALL_{AFN}} = \{< A > |\text{A es un AFN y L(A)} = \Sigma^*\}$
$\overline{ALL_{AFN}}$ N = "Con entrada $< A >$ donde A es un AFN:

1. Marca el estado inicial del AFN
2. Repite 2^q veces donde q es el número de estados de A
3. De forma no determinista, elige un símbolo de entrada y cambia las posiciones de las marcas en los estados de A para reflejar la transición no determinista, generada por la lectura del símbolo.
4. Acepta si en la etapa 3 en algún punto ninguna marca está en un estado de aceptación, i.e., si N encontró una cadena que A rechace. De otro modo *rechaza*"

El número de configuraciones de símbolos marcados es igual a $|2^q| = 2^q$
No se sabe si está en NP o co-NP
Si M corre con espacio f(n) ¿Cómo acotamos su tiempo de ejecución?
M corre en tiempo $2^{\mathcal{O}(f(n))}$

SPACE(foo) = {L — existe una MT que corre en espacio f(n) y decide a L}
NSPACE(f(n)) se realiza con máquinas de turing no deterministas
PSPACE = $\cup_{k\in\mathbb{N}} SPACE(n^k)$
NPSPACE = $\cup_{k\in\mathbb{N}} NSPACE(n^k)$
¿PSPACE = NPSPACE?
EXPTIME = $\cup_{k\in\mathbb{N}} TIME(2^{n^k})$
PSPACE $\subseteq$ EXPTIME
N corre en espacio f(n)
Teorema (SAVITCH) Para cualquier función $f : \mathbb{N} \to \mathbb{R}$, con $f(n) \geq n$,
$NSPACE(f(n)) \subseteq SPACE(f^2(n))$
Por tanto PSPACE = NPSPACE
Alcanzabilidad: N, c_1, c_2 configuraciones y t entero positivo.
Q: ¿Se puede alcanzar c_2 desde c_1 en a lo más t pasos siguiendo las reglas de N?
H $\to$ ALCANZA
N, w
¿N acepta a w?
C_{start} = configuración inicial
$C_{accept} = q_a\sqcup$
ALCANZA(C_{start}, C_{accepr}, t)
$t = 2^{\mathcal{O}(f(n))}$
$logt = \mathcal{O}(f(n))$
La pila no crece más de log y en cada llamada recursiva se guarda espacio f(n), de ahí que el algoritmo corre en $\mathcal{O}(f^2(n))$
Dem: Sea N una MTN de espacio f(n)
Construimos una MTD M "que simula a N". M usa la subrutina ALCANZA que toma 3 argumentos, dos configuraciones c_1 y c_2 y un entero positivo t y ALCANZA(c_1, c_2, t) acepta si N en t pasos puede alcanzar la configuración c_2 partiendo de c_1. En otro caso rechaza. Para facilitar el análisis supongamos que t es una potencia de 2.
ALCANZA= "Con entrada c_1, c_2, t:

1. Si t= 1, verifica si $c_1 = c_2$ o si tras un paso en la ejecución de N, c_1 alcanza a c_2. *Acepta* si se cumple alguna de las dos condiciones y *rechaza* de otro modo
2. Por cada configuración c_m de N que usa espacio f(n):
3. Calcula ALCANZA($c_1, c_m, \frac{t}{2}$)
4. Calcula ALCANZA($c_m, c_2, \frac{t}{2}$)
5. Si 3 Y 4 aceptan, *acepta*
6. *Rechaza*"

Necesitamos modificar N para que tenga una única configuración de aceptación: al aceptar, borra su cinta y regresa el cabezal a la posición más a la izquierda y se mueve a un nuevo estado de aceptación, esta configuración es C_0.
Elegimos un entero d tal que N no tenga más de $2^{f(n)}$ es una cota superior por el tiempo de ejecución de cualquier número con cualquier entrada.
Entonces, basta con proponer a M M="Con entrada w:

1. Responde $ALCANZA(C_3, C_0, 2^{f(n)})$

Como ALCANZA resuelve alcanzabilidad, M "simula" a N.
Al llamarse recursivamente, ALCANZA guarda los valores de C_1, C_2 y t en la pila, para poder restaurarlos al volver de la llamada. Cada nivel de la recursión usa $\mathcal{O}(f(n))$ espacio adicional. Además, cada nivel divide t por 2, es inicialmente $2^{df(n)}$, por lo que la profundidad de la recursión es $\mathcal{O}(log 2^{df(n)}) = \mathcal{O}(f(n))$. Por tanto el espacio total que se utiliza es $\mathcal{O}(f^2(n))$
¿Cómo obtenemos f(n)?
Probamos $f(n) = 1, f(n) = 2, f(n) = 3$, ..,
¿Cuándo nos detenemos?
Si $f(n) = i$, ¿podemos alargar alguna configuración de longitud i+1 ?
En cada nivel se guarda $\mathcal{O}(f(n))$
$P \subseteq PSPACE$, $PSPACE \subseteq EXPTIME$
$NP \subseteq NPSPACE$ $P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$
Corolario : PSPACE = NPSACE
Dem: El cuadrado de un polinomio es un polinomio ■
$P \subseteq NP \subseteq PSPACE = NSPACE \subseteq EXPTIME$
Def Un lenguaje B es PSPACE-completo si

- B ∈ PSPACE
- Para cada A en PSPACE $A \leq_P B$ (A se puede reducir en tiempo polinomial a B)

Forma Normal Prenex (todos sus cuantificadores estén al inicio)
Fórmulas Booleanas Totalmente Cuantificadas en FNP
TQBF = $\{< \varphi > | \varphi$ es una fórmula booleana totalmente cuantificada verdadera$\}$
Teo TQBF es PSPACE-completo
Dem: Proponemos a T una MT tal que
T: "Con entrada $< \varphi >$, donde φ es una f.b.t.c:

1. Si φ no tiene cuantificadores, entonces solo tienes constantes. Evalúa Φ y *acepta* si es verdadera y *rechaza* si no.
2. Si $\varphi = \exists x \Psi$ llamamos a T con Ψ primero sustituyendo cada ocurrencia de x en Ψ por 0 y después por 1. Si alguno de las dos acepta, *acepta*.
3. Si $\varphi = \forall x \Psi$, llama a T con Ψ primero sustituyendo cada ocurrencia de x en ψ por 0 y después por 1. Si las dos aceptan,*acepta*, si no *rechaza*.

El número de llamadas recursivas es $\mathcal{O}(n)$ donde n es el número de cuantificadores en φ, y en cada llamada sólo necesitamos usar una cantidad constante de espacio. Por lo tanto T corre en espacio lineal respecto a la longitud de φ.
Luego TQBF ∈ PSPACE
Sea A ∈ TQBF, sea M una MT de espacio polinomial que decide a A. Para cada entrada de A, vamos a construit una f.b.t.c. φ tal que φ sea verdadera si y sólo si M acepta a w.
Sean c_1 y c_2 conjuntos de variables que representan configuraciones, es decir, por ejemplo $c_1 = \{x_1, ..., x_l\}$ donde $l = f(n)$ cada x_i es el contenido de una celda en la configuración de la MT M. Dado un entero t construimos $\varphi_{c_1,c_2,t}$ una fórmula que es verdadera si y sólo si la configuración c_1 alcanza a c_2 en t pasos o números.
Si c_0 es la configuración inicial y c_d es la configuración de aceptación, entonces $\varphi_{c_0,c_d,h}$ es verdadera si y sólo si M acepta a c_d donde $h = 2^{df(n)}$ donde d es una constante suficientemente grande tal que M no corre por más de h pasos. Cuando t=1 $\varphi_{c_0,c_d,h}$ φ considera que o bien $c_1 = c_2$, o c_1 alarga a c_2 en una transición- En el segundo , utilizamos la fórmula del Teorema de Cook para las ventanas suponiendo que c_1 y c_2 son configuraciones adyacentes.
Si $t > 1$ $\varphi : c_1, c_2, t = \exists m_1 [\varphi_{c_1,m_1,\frac{t}{2}} \& \varphi_{m_1,c_2,\frac{t}{2}}]$

En su lugar proponemos
$\varphi : c_1, c_2, t = \exists m_1 \forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\} \varphi_{c_3,c_4,t}$ Gracias al cuantificador universal tenemos los valores
$\varphi_{c_1,m_1,\frac{1}{2}}$ y $\varphi_{m_1,c_2,\frac{1}{2}}$
El tamaño de $\varphi_{c_1,c_2,h}$ con $h = 2^{d(f(n))}$. Por cada llamada recursiva se usa espacio lineal y la profundidad $\log 2^d f(n) \in \mathcal{O}(f(n))$
La longitud de $\varphi_{c_1,c_2,h}$ es $\mathcal{O}(f^2(n))$ Como $f(n) = n^t$ para algún entero k entonces $\mathcal{O}(f^2(n)) = \mathcal{O}(n^{2t})$ que es polinomial.
Sea $\varphi = \forall x_1, \exists x_2, \forall x_3, ...Qx_k[x]$ $Q \in \{\exists, \forall\}$
$\varphi_1 = \exists x_1, \forall x_2, \exists x_3((x_i \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3}))$
$\varphi_2 = \exists x_1, \forall x_2, \exists x_3((x_i \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3}))$
FORMULA-GAME $= \{< \varphi > \mid$ El jugador E tiene una estrategia ganadora para el juego asociado a $\varphi\}$
Teo FORMULA-GAME es PSPACE-complete
TBQF $= \{< \varphi > |\varphi$ es una fórmula boolena totalmente cuantificada en FNP, verdadera $\}$
Dem: Demostraremos que FORMULA-GAME = TQBF
Si φ es verdadera, entonces por cada elección que haga A, existe una elección para E que hace a φ verdadera. Luego, E tiene una estrategia ganadora. Recíprocamente si E tiene una estrategia ganadora, entonces existe una asignación para las variables cuantificadas existencialmente par cualquier elección de las variables cuantificados universalmente que hacen a φ verdadera. Luego FORMULA-GAME $\subseteq$ TQBF ■
Geografía-Generalizado: Se juega sobre una digrpafica D y empezando en un vértice distinguido b. Los jugadores 1 y 2 alternan turnos eligiendo un vértices en la exvecindad del vértice actual. No se pueden repetir vértices y el primer jugador que no puede elegir un vértice pierde.
GG $= \{< D, b > \mid$ El jugador I tiene una estrategia ganadora en GG sobre D empezando en b$\}$
Teo GG es PSPACE-completo
M = "Con entrada ¡D,d¿ con D una digráfica y b un vértice de D:

1. Si b tiene exgrado cero <u>*rechaza*</u> (I no tiene movimientos)
2. Borra b y todas las flechas incidentes en b para obtener G_1
3. Para cada $v \in V^+(b)$ (exvecindad de b), llama recursivamente a M con entrada $< G_1, v >$
4. SI todas las llamadas en 3 aceptan, entonces <u>*rechaza*</u> (II tiene una estrategia ganadora). Si no, II no tiene estrategia ganadora, por lo que I si la tiene, <u>*acepta*</u> "

Notamos que GG es decidido por M. Además, M sólo necesita guardar un vértice en la pila de recursión en casa llamada. Se sigue que M corre en espacio lineal, por tanto, GG $\in$ SPACE
$\varphi = \exists x_1 \forall x_2 \exists x_3((x_1 \vee x_2) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3}))$
Para cada variable x_i, en el orden en el que aparecen cuantificadas en φ, creamos un "rombo".
Para dos variables consecutivas unimos a sus "rombos" con una flecha del vértice inferior del primero al superior del segundo, por ejemplo si en φ aparecen $\forall x : \exists x_{i+1}$ unimos el rombo de x_i con el de x_{i+1}.
Para la i-ésima clausula $(x_1 \vee x_e \vee ... \vee x_k)$ creamos la siguiente estructura: Un vértice por cada literal y un vértice adicional que los domina a todos.
Para cada vértice correspondiente a una literal en los gadgets anteriores agregamos una flecha hacia la copia de la literal correspondiente en los "rombos" construidos anteriormente. El vértice b es el vértice superior del primer "rombo" y creamos un nuevo vértices c que tiene flecha hacia el vértice dominante en el gadget de cada clausula y recibe flecha desde el vértice inferior del último "rombo".
Obs. Una vez que I tira en el vértice superior de un "rombo", los siguientes dos movimientos están forzados y la siguiente decisión la toma II en el vértice superior del siguiente rombo. Esto alternará hasta que se acaben los "rombos". Podemos suponer que I empieza y termina, y que las decisiones

antes descritas son alternadas, quizá agregando cuantificadores a φ con fórmulas que no ocurran en Ψ.
Como I toma la decisión sobre el último "rombo", tras dos tiros forzados, II elige sobre que clausula tirar. Sin importar cual elija, si E tiene una estrategia ganadora, podemos garantizar que todas las clausulas son verdaderas, por lo que I puede elegir la literal verdadera en su siguiente tiro y como esa literal se usó en el "rombo" correspondiente II no tiene movimientos.
Si I tiene una estrategia ganadora, entonces por la observación anterior, sin importar que clausula elija II al final del juego, I siempre puede elegir una literal en esa clausula por la que se paso previamente en el camino generado por el juego. Por tanto, para cada decisión de I hacia la izquierda, E elige I como el valor para la variable correspondiente y por cada decisión de I hacia la derecha E elige 0 como el valor de la variable correspondiente. Esto garantiza que cada clausula tiene al menos una literal verdadera. ■

3.2 Espacio sublineal

Hasta el momento la cantidad de espacio usado $f(n)$ cumplía que $f(n) > n$. Para cada espacio subliminal necesitamos modificar un modelo. Vamos a correr una MT con dos cintas una de sólo-lectura (entrada) y una de lectura-escritura (trabajo). En la cinta de trabajo solo se puede ocupar $\mathcal{O}(logn)$ espacio. El cabezal de la cinta de entrada no puede salir del espacio que ocupa la entrada, por lo que esta MT debe tener algún mecanismo para saber cuando el cabezal de la primera cinta está en la posición más a la izquierda o al final de la entrada.
L = SPACE(logn)
Los lenguajes que pueden ser decididos por una MTD en espacio logarítmico.
NL = NSPACE(logn)
Los transductores son máquinas que tienen una cinta de solo salida y que su fin ultimo es generar una salida a diferencia de las MT cuyo fin es reconocer un lenguaje (aceptar o rechazar)
Def Transductor de Espacio Logarítmico. Es una MT con tres cintas, una será de sólo-lectura (entrada) con la misma restricción de que e cabezal no puede moverse del inicio de la entrada. Va a tener una cinta de lectura-escritura (trabajo) que tendrá espacio logarítmico. Y va a tener una cinta de sólo-escritura (salida)
Computa $f : \Sigma^* \to \Sigma^*$ con salida $f(w)$ y siempre que empiece con w en su cinta de entrada, la ejecución sólo usa espacio O(logn) en la cinta de trabajo y termina su ejecución con $f(w)$ en la cinta de salida. Si A y B son lenguajes decimos que A es reducible en espacio logarítmico a B. $A \leq_L B$ si hay una función computable f, calculada por un transductor de espacio logarítmico tal que $w \in A$ si y sólo si $f(w) \in AS$
Un lenguaje A es NL-completo si

- $A \in NL$
- Para cada B$\in$ NL, B$\leq_L$ A

Tenemos al lenguaje $\{0^k1^k | k \in \mathbb{N}\} = A$ $A \in L$
Veamos que tendrá espacio logarítmico:
M = " Con entrada w

1. Revisa que w sea de la forma 0*1*. De no ser así, <u>rechaza</u>. $\mathcal{O}(1)$
2. Cuenta los 0 en w y escribe cuántos hay en un contador en la cinta de trabajo. $\mathcal{O}(logn)$
3. Cuenta los 1 en w y escribe cuántos hay en la cinta de trabajo. $\mathcal{O}(logn)$
4. Si los contadores son iguales, <u>acepta</u>. Si no, <u>rechaza</u>. $\mathcal{O}(1)$

Una MT que corre en espacio $f(n)$ usa tiempo a lo más $2^{O(f(n))}$ $f(n) \geq n$
Dada una MT de es espacio logarítmico M y una entrada w, una configuración de M sobre w es una elección de un estado contenido de la cinta de trabajo y posiciones de los dos cabezales
Si M corre en espacio $f(n)$ y w es una cadena de longitud de n, entonces M tiene el siguiente número de configuraciones sobre w:
$|Q| = q$ (estados) $|p^k| = g$ (cardinalidad de los símbolos en el alfabeto)
$nqg^{f(n)}f(n) = n2^{\mathcal{O}(f(n))}$
$f(n) \geq log\ 2^{\mathcal{O}(f(n))}\ 2^{\mathcal{O}(logn)}\ 2^{klogn} = 2^{logn^k} = n^k =$ polinomial
$n2^{\mathcal{O}(f(n))}$ podemos elegir una constante suficientemente grande tal que $n2^{\mathcal{O}(f(n))}$ sea $2^{\mathcal{O}(f(n))}$
$g^{(f(n))} \leq 2^k$
Si M corre en espacio logarítmico entonces corre en tiempo polinomial.
En la demostración del teorema de Savich $log(n2^{\mathcal{O}(f(n))}) = log(n) + log2^{\mathcal{O}(f(n))} = logn + \mathcal{O}(f(n)) = \mathcal{O}(f(n))$.
y en cada llamada recursiva sólo necesitamos espacio $\mathcal{O}(f(n))$ para guardar la configuración de M en la pila.
¿NL?
$PATH = \{< D, s, f > | D$ es una digráfica con una st-trayectoria$\}$
Eliges un camino de forma no determinista
M = "Con entrada $< D, s, t >$ con D una digráfica, $s, t \in V$:

1. Inicializa un contador c en la cantidad de vértices de la digráfica y una variable x en S, en la cinta de trabajo.
2. Mientras $c > 0$:
3. Si $x = t$, *acepta*.
4. Elige un vértice v de forma no determinista en la exvecindad de x y reemplaza x por v.
5. Decrementa c en una unidad.
6. *Rechaza* "

Espacio: $2logn = \mathcal{O}(logn)$
Teo PATH es NL-completo
Dem: Sea $A \in NL$ y supongamos que N es una MTN que decide a A. Sea w una cadena. Construyamos una digráfica D tal que cada vértice de D es una configuración de N sobre w, y donde $(c_i, c_j) \in A_D$ si y sólo si hay una transición wN de c_i o c_j. Para generar la lista de vértices de D vamos a generar todas las cadenas de longitud a lo más grande de N y vamos a verificar si cada una es o no una configuración válida de N. Si no lo es, continuamos, si sí, la copiamos a la cinta de salida.
Las flechas se generan análogamente pero probando parejas de configuraciones de N.
El vértice s es la configuración inicial c_s y para t podemos modificar a N como en el teorema de Savitch para que haya una única configuración de aceptación c_a.Entonces t es c_a. Ya tenemos $< D, s, t >$. Si N acepta a w, entonces alguna rama de N termina un c_a. Los nodos de dicha rama conforman una st-trayectoria en D. Recíprocamente, si hay una st-trayectoria en D, entonces hay una sucesión de transiciones de N que empezando en c_s terminan en c_a. Luego, alguna rama del árbol de ejecución de N termina en la configuración de aceptación y por lo tanto N acepta a w.
Como en la cinta de trabajo hay a lo más dos configuraciones de N, el espacio utilizado es $\mathcal{O}(logn)$■.
Teo Si A y B son lenguajes tales que $A \leq_L B$ y $B \in L$ entonces $A \in L$
M = "Con entrada w

1. Calcula f(w)

2. Corre $M_B(f(w))$ y responde lo mismo"

Dem: Sea M_A el transductor que calcula f(w) con espacio logarítmico y M_B la MT de espacio logarítmico que decide a B, creamos una MT M de espacio logarítmico que haga lo siguiente: $SimuleaM_B$, cada que necesite al siguiente símbolo para continuar su ejecución va a escribir en un contador la posición de la entrada que necesita. En ese punto M guarda el contenido de la cinta de trabajo de M_B y ejecuta a M_A hasta generar la entrada necesaria de f(w). Cada vez que haga esto, borra el trabajo anterior y reinicia la ejecución de M_A desde cero. Para simular a M_B se usa espacio $\mathcal{O}(logn)$ al igual que para simular a M_A. Además necesitamos a una celda para guardar el símbolo que está guardando M_B y espacio log. para cada contador. En total, el espacio utilizado es $\mathcal{O}(logn)$

Corolario Si un lenguaje NL-Completo está en L, entonces L = NL

$bfTeo$ NL = coNL

Dem: PATH es NL-completo $\overline{PATH} \in NL$

C= Número de vértices que se alcanzan desde s

Por cada vértice v de D, adivina si v es alcanzado o no por s, y en caso positivo lo verifica. Si la verificación falla rechaza. Si no, aumenta un contador que cuenta cuantos vértices se alcanzan desde s en 1. Cuando ya recorrió todos los vértices compara el contador contra c. Si son distintos, rechaza. Si en algún momento se verifica que s alcanza a t, $\underline{rechaza}$. Si son iguales, entonces M encontró a todos los vértices que se alcanzan desde s y t no es uno de ellos, entonces $\underline{acepta}$.

Necesita una variable para hace la verificación, una para verifica y el contador en espacio log, por lo tanto toma espacio $\mathcal{O}(logn)$

$A_i = \{$vértices en D que se alcanzan desde s en una distancia igual a i$\}$ $A_0 = \{s\}$ $c_0 = 1$

A_{i+1} usando A_i y $c_i = \{A_i\}$

Verificamos que para cada $v \in V$, si $v \in A_{i+1}$

Para cada vértice v de V adivina si está o no en A_i. i sí, lo verifica, aumentando un contador, y busca si (u,v) está en las flechas de D, y aumentamos un contador que cuenta los vértices de A_{i+1} (queremos encontrar i+1). Si $(u, v) \notin A_d$, continua verificando a los demás vértices. Cuando todos los vértices han sido verificados comparamos a d con c_i, si son iguales, v no está en A_{i+1}