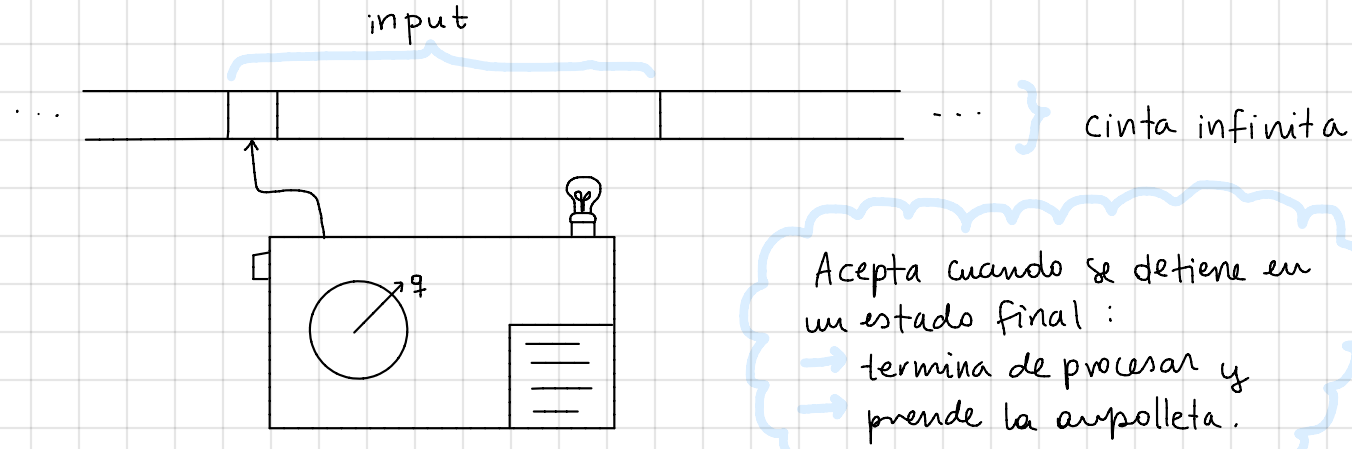
Máquinas de Turing (MT)

11/06/19



Acepta cuando se detiene en un estado final:
- termina de procesar y
- prende la ampolleta.

- Transiciones se ven algo así:

símbolo a leer	a	$\longrightarrow$ a'	lo que se escribe
		q'	nuevo estado
estado actual	q	dirección	hacia dónde se mueve el cabezal

- La máquina se define como:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

- Q = conj. de estados.
- Σ = alfabeto del input
- Γ = alfabeto de la cinta $(\Sigma \subseteq \Gamma)$
- q_0 = estado inicial $(q_0 \in Q)$
- B = "símbolo blanco" $(B \in \Gamma,\ B \notin \Sigma)$
- F = Conj. de estados finales $(F \subseteq Q)$
- $\delta: Q \times \Gamma \nrightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$

δ puede estar no definido para alguna configuración.

- Rechaza si:
  - no se detiene o
  - se detiene en un estado no final

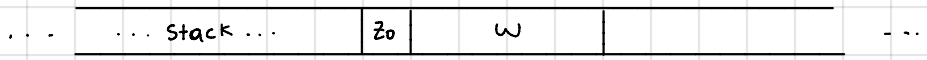
- Ejemplo: 1 $L = \{0^i 1^i \mid i \geq 1\}$

	0	1	X	Y	B
q_0	$(X, q_1, \rightarrow)$	-	-	$(Y, q_3, \rightarrow)$	-
q_1	$(0, q_1, \rightarrow)$	$(Y, q_2, \leftarrow)$	-	$(Y, q_1, \rightarrow)$	-
q_2	$(0, q_2, \leftarrow)$	-	$(X, q_0, \rightarrow)$	$(Y, q_2, \leftarrow)$	-
q_3	-	-	-	$(Y, q_3, \rightarrow)$	$(B, q_4, \leftarrow)$
(q_4)	-	-	-	-	-

2 $L = \{ 0^i 1^i 2^i \mid i \geq 1 \}$

Se puede hacer similar a lo anterior.

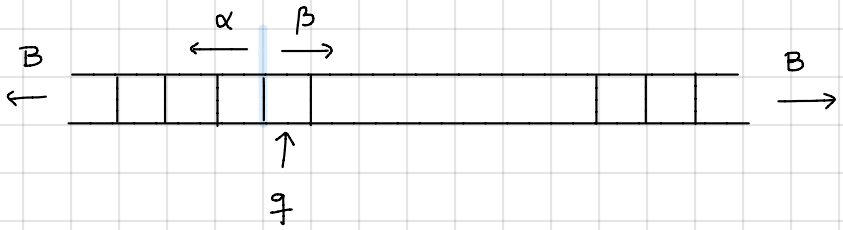
- Las máquinas de Turing aceptan más lenguajes que los AA. Esto se ve suponiendo que en la cinta se pone un stack, que puede comportarse igual que el autómata:



y puedo hacer todos los stacks que quiera.

- ¿Qué no acepta? Por ej, $L = \{ \langle G \rangle \mid$ son ambiguos $\}$.

- Descripción Instantánea (DI):



→ $\alpha q \beta$; $\beta, \alpha \in \Gamma^*$, $q \in Q$ // Inicial: $q_0 w$.

- Computación: $\alpha, \beta \in \Gamma^*$, $a, b, c \in \Gamma$

→ $\alpha a q b \beta \vdash_M \alpha a c p \beta$ ssi $\delta(q, b) = (p, c, \rightarrow)$

→ $\alpha a q b \beta \vdash_M \alpha p a c \beta$ ssi $\delta(q, b) = (p, c, \leftarrow)$

$\vdash_M$: Un paso

- Lenguaje aceptado: $\vdash_M^*$: Varios pasos.

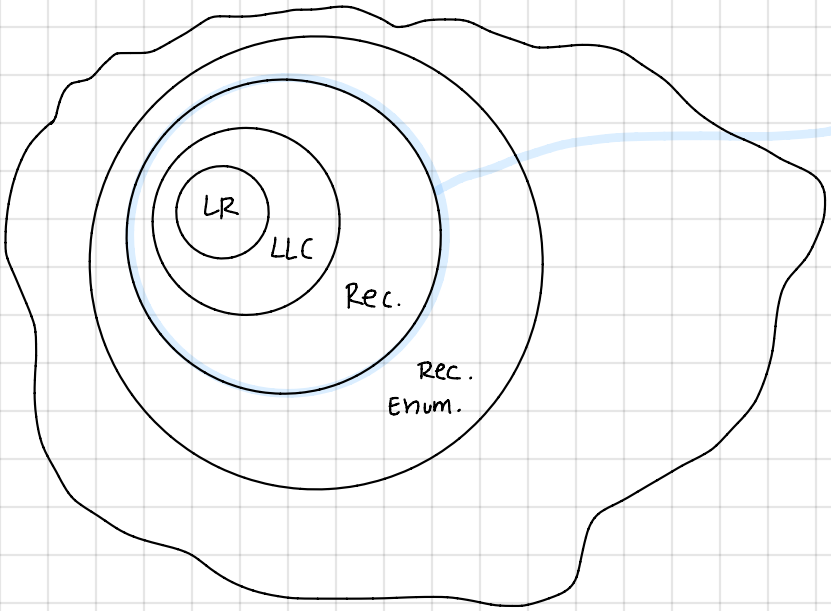
→ $L(M) = \{ w \in \Sigma^* \mid q_0 w \vdash_M^* \alpha p \beta$, $p \in F$ y no hay transición definida para p en el 1er símbolo de $\beta \}$.

- Def. L es recursivamente enumerable si existe una MT M tq $L = L(M)$.

- Def. L es recursivo si existe M que siempre se detiene tq $L = L(M)$.

- Def. Un algoritmo es una MT que se detiene para todos sus inputs.

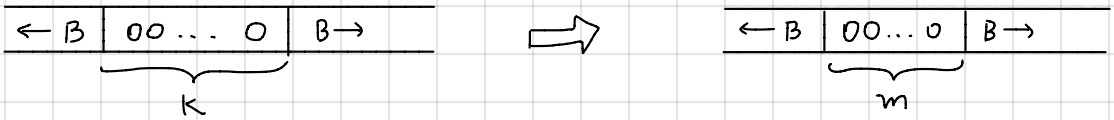
- LLC son subconjuntos de los lenguajes recursivos:



Problemas dentro de esto se llaman decidibles
Hacia afuera son indecidibles.

- Podemos ver las MT como funciones:

→ $M \Longleftrightarrow f : \mathbb{N} \to \mathbb{N}$



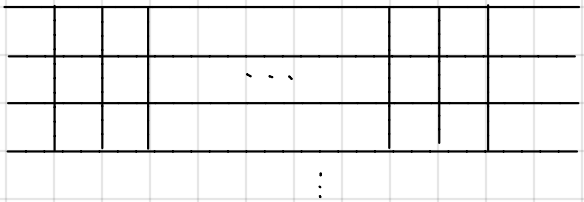
$f(k) = m$

→ $M \Longleftrightarrow \mathbb{N}^2 \to \mathbb{N}$; ¿$f(k, l)$? : $0^k 1 0^l$

$k+l$ => $0^k 1 0^l \vdash_M^* 0^{k+l}$

Técnicas de Descripción de MT

- Mantener memoria finita → Control Finito: guardar en el estado lo que necesitemos, $[q, -, -, -]$ (con memoria finita).
- Múltiples "pistas" → se puede pensar que la cinta guarda más de 1 símbolo:



Pista 1
Pista 2
Pista 3
⋮

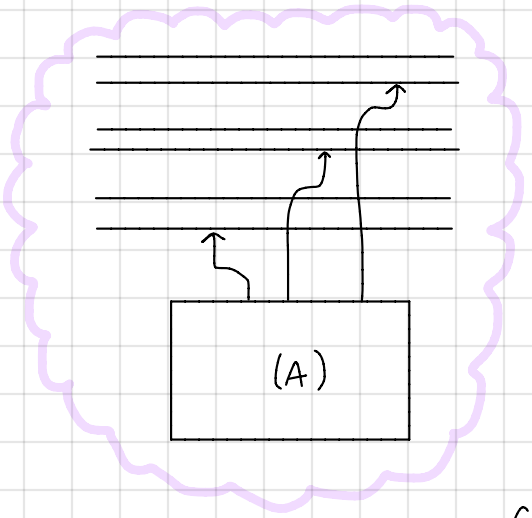
Un símbolo: $\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$

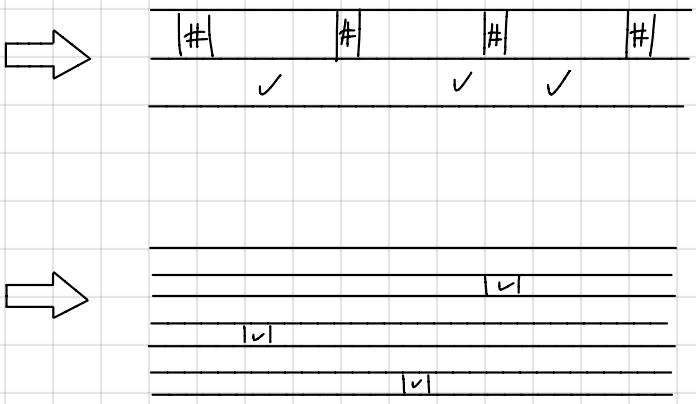
- Marcar símbolos
- Shiftear símbolos
- Copiar parte del string.
- Llamar a subrutinas.

13/06/19

- Hipótesis de Church-Turing:
Todo lo computable es lo que puede hacer una máquina de Turing. (¿Por qué no es teorema? Lo computable no está tan bien definido, por lo que no se puede demostrar actualmente).

- Supongamos que tenemos una máquina con 3 cintas (y 3 cabezales), esta puede ser más rápida (hacer menos transiciones), pero es equivalente a una máquina de Turing. ¿Cómo simularla?





Control finito: $[q, \_, \_, \sqcup_{c3}, \sqcup_{c1}, \sqcup_{c2}]$

Cant. de pistas a la izq

Cant. de pistas a la der.

Barre a la izq. mirando donde están los cabezales y la info, y a la der. ejecutando

→ Si (A) hace N transiciones, esta hace $O(N^2)$, porque máxima separación de cabezales es $2N$
$\Rightarrow 2+4+6 \dots + 2N \in O(N^2)$.

- Con esto concluimos que siempre podemos simular una máquina como (A) como una MT, con tiempo cuadrático, independiente de su cant. de pistas.

Máquina de Turing No Determinista

- $M = (Q, \Sigma, \delta, q_0, B, F)$

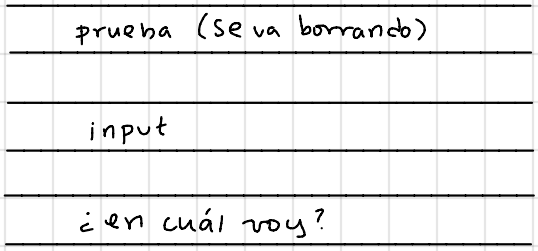
→ Det: $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\rightarrow, \leftarrow\}$

→ No det: $\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{\rightarrow, \leftarrow\}}$

Subconjunto Finito.

- Teo: Toda MTND puede ser simulada por una MTD (pero muy lento).

– ¿Cómo hacerlo? 3 cintas:



Cualquier string de símbolos entre 0 y, máximas opciones de transición (M) − 1, por ejemplo:

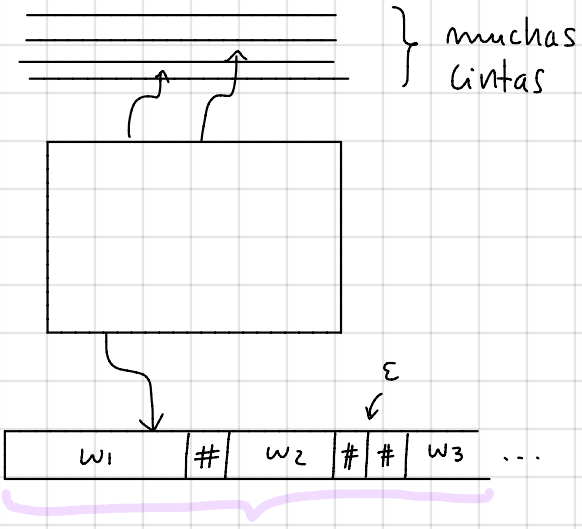
3 7 1 (M-1) 4, y se le suma 1 al nº total.

Si llega a una configuración que da un estado final, acepta el string, si no, se queda pegada.

¿Tiempo? Si MTND hizo N transiciones, con máx. 2 opciones por transición => $O(2^N)$ para MTD.

! No se sabe si lo anterior se puede hacer más rápido (o si esto es lo más rápido).

Máquinas Enumeradoras



Cinta de output: $\Sigma \cup \{\#\}$

Máquinas generadoras son deterministas.

- Cinta de output:
  - Solo escribe
  - Solo se mueve hacia la derecha
  - Los w_i forman el lenguaje generado por la máquina: $G(M)$
  - Nos gustaría que no terminara nunca, si tiene finitas transiciones, el lenguaje es finito $\Rightarrow$ es regular.

- Para cualquier L_1 LR, L_2 LLC existen M y M' tq $L_1 = G(M)$ y $L_2 = G(M')$, y se generan en orden.

- L rec. enum., $L = L(M)$. ¿$\exists M'$ tq $L = G(M')$? Se puede con mucho cuidado, dándoles tiempo de ejecución (cant. de transiciones) limitado a c/ string, sumándoles más tiempo mientras algunos aceptan. Lo que voy guardando son los strings que no han sido aceptados y el tiempo de ejecución actual. De esto nace su nombre: son rec. enum. porque pueden ser generados uno por uno, por una máquina.

- Una máq. generadora genera el lenguaje en orden ssi el lenguaje es recursivo.

- Si una máq. genera un leng $\Rightarrow$ el leng. es rec. enum. (Dem.)

Máquina Universal de Turing / Indecidibilidad

18/06/19

- $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

→ Suposiciones sobre M (válidas para toda MT):

- $M = (\{q_1, \dots, q_N\}, \{0,1\}, \{0,1,B\}, \delta, q_1, B, \{q_2\})$ } MT normalizada.

  - $0 \equiv a_1$
  - $1 \equiv a_2$
  - $B \equiv a_3$
  - $\leftarrow \equiv d_1$
  - $\rightarrow \equiv d_2$
  - $\delta(q_i, a_j) = (q_k, a_\ell, d_m)$ → La MT solo está determinada por su δ, y c/ transición está definida por sus i, j, k, ℓ y m.

    |||

  - $0^i 1 0^j 1 0^k 1 0^\ell 1 0^m$ → Define por completo una transición.

  - $M \equiv 111 \_\_\_\_ 11 \_\_\_\_ 11 \_\_\_\_ \dots \_\_\_\_ 111 = \langle M \rangle$

    String de transición. — Codificación de una MT

- Cant. de MT es infinita numerable, y no es más que el nº de strings.

- Todo string que no toma la forma anterior se dirá que define una máquina M1 que acepta el lenguaje vacío: $L(M_1) = \phi$, por simplicidad.

-

Σ^* / MTs	w_1	w_2	w_3	w_4	w_5	...	
$\langle M_1 \rangle$	1	1	0	0	1	1	← $L(M_1)$
$\langle M_2 \rangle$	0	0	0	0	1	0	← $L(M_2)$
$\langle M_3 \rangle$	1	0	1	0	1	0	← $L(M_3)$
$\langle M_4 \rangle$	1	1	1	1	1	1	← $L(M_4)$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

1 si la máq. acepta el string, 0 si no.

- Todos los lenguajes recursivamente enumerables aparecen en la matriz anterior.

- Si encuentro una fila que no pertenece a la matriz, entonces no es un lenguaje rec. en.

Ahora ¿cómo construimos / encontramos lenguajes que no están en las filas de la matriz? Por argumento diagonal, definimos:

Fila ☆	0	1	0	0	...	← L_D (leng. diagonal)

L_D no es rec. en. porque no pertenece a la matriz. Este lenguaje es un conjunto enumerable pero que no puede ser ordenado por un computador (mecánicamente).

IMPORTANTE!

→ $L_D = \{ w_i \mid w_i \notin L(M_i) \}$

$= \{ \langle M_i \rangle \mid \langle M_i \rangle \notin L(M_i) \}$

$\iff$ $L_D = \{ \langle M \rangle \mid \langle M \rangle \notin L(M) \}$

Por demostrar que L_D no es rec. enum. Por contradicción:

Supongamos que existe M_D tq tq $L_D = L(M_D)$

¿ $\langle M_D \rangle \in L(M_D)$?

→ Si $\langle M_D \rangle \in L(M_D) \Rightarrow \langle M_D \rangle \in L_D \Rightarrow \langle M_D \rangle \notin L(M_D)$

→ Si $\langle M_D \rangle \notin L(M_D) \Rightarrow \langle M_D \rangle \in L_D \Rightarrow \langle M_D \rangle \in L(M_D)$

$\Rightarrow\Leftarrow$

∴ L_D no es rec. enum. $\Rightarrow$ no es aceptado por una MT.

Acabamos de demostrar que existe al menos un lenguaje no rec. enum., es decir, ese espacio que no conocíamos no es vacío. Aún no demostramos que existe algo entre los leng. rec. y los no rec. enum.

Lenguaje Universal

$L_U = \{ \langle M \rangle w \mid w \in L(M) \}$ → Contiene el funcionamiento de todas las MT's y todos los strings que aceptan.

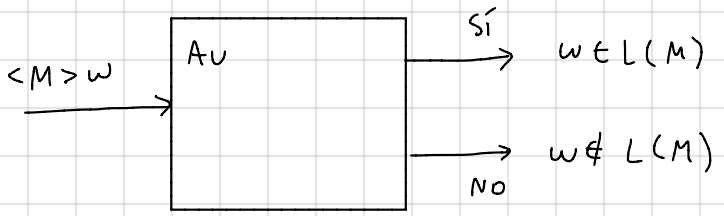
¿ Máquina Universal M_U, tq $L_U = L(M_U)$? Su existencia implica que no se necesita construir ninguna otra máquina. Y existe! Nuestro computador, celular, etc. Se demuestra por construcción de M_U.

→ Como existe $M_U \Rightarrow L_U$ es rec. enum. ¿ es L_U recursivo?

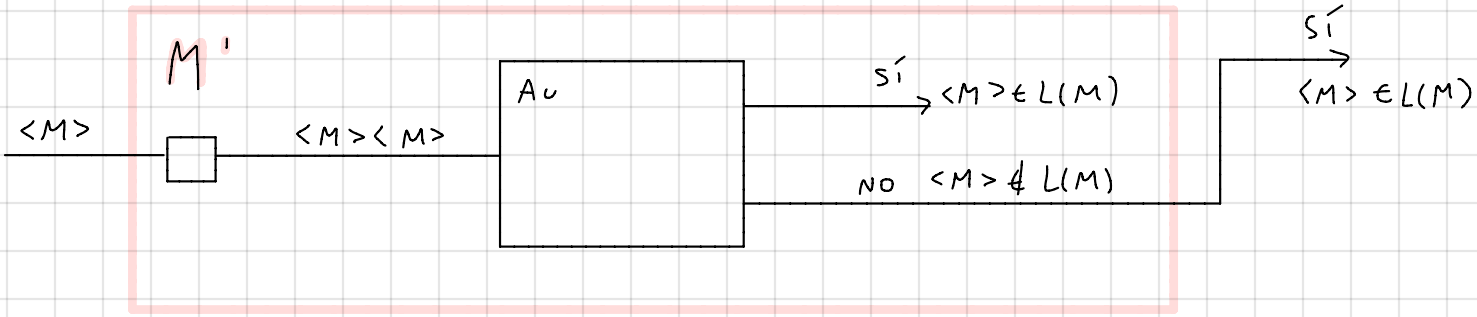
PD. L_U no es recursivo.

Supongamos L_U recursivo $\Rightarrow$ existe un algoritmo para L_U

Digamos que tiene esta forma:



Si $w = \langle M \rangle$ podemos tomar el algoritmo y agregar una máquina M':

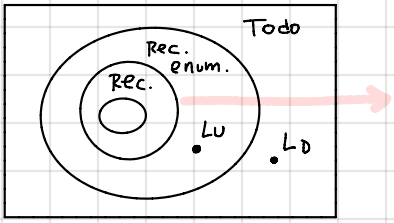


Lo anterior implica $L(M') = L_D$, pero L_D no es rec. enum. así que no hay máquina que la genere. $\Rightarrow\Leftarrow$

$\therefore$ A_U no puede existir $\Rightarrow$ L_U no puede ser recursivo.

Conclusión: si existiera un algoritmo para L_U, existiría una máquina para L_D. La demostración se hace por reducción, y concluye que L_U es almenos más difícil que L_D.

- Lo que encontramos hoy:



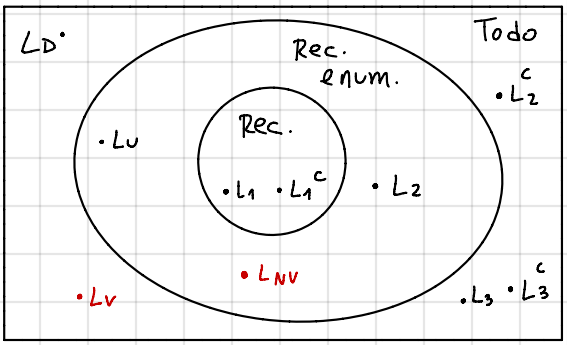
Desde aquí todo es indecidible.

- L_U es indecidible y L_D también. L_D es más indecidible que L_U.
- Consideremos estos lenguajes:
  - $L_{NV} = \{ \langle M \rangle \mid L(M) \neq \phi \}$
  - $L_V = \{ \langle M \rangle \mid L(M) = \phi \}$

  ⇝ Es un poco más "fácil", pero hay que simularla con cuidado (por si se queda pegada en un string)

→ L_{NV} es rec. enum.

- Si un leng. es recursivo ⇒ su complemento también lo es (porque termina para aceptar y rechazar).
- Únicas opciones:



* Por demostrar.

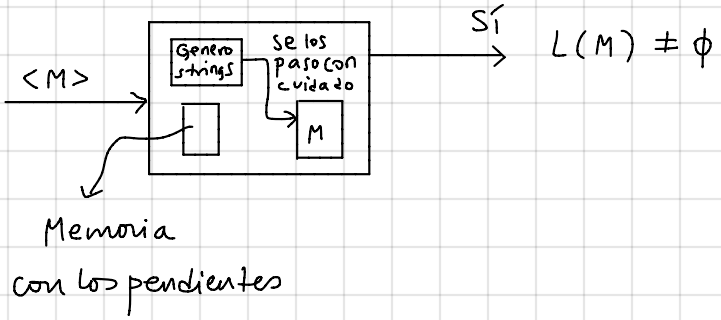
20/06/19

- Siempre hablaremos de problemas refiriéndonos a lenguajes, porque todo problema se puede escribir como un lenguaje para aceptar por una máquina.

Más lenguajes / problemas indecidibles.

- $L_{NV} = \{ \langle M \rangle \mid L(M) \neq \phi \}$

"Dem". fs rec. enum.

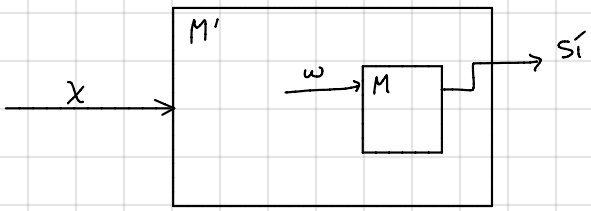


¿Es recursivo? No lo es. Dem. por reducción

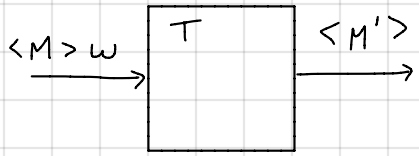
PDQ L_{NV} no es recursivo.

$\Leftrightarrow$ PDQ si L_{NV} fuera recursivo entonces L_U también sería recursivo.

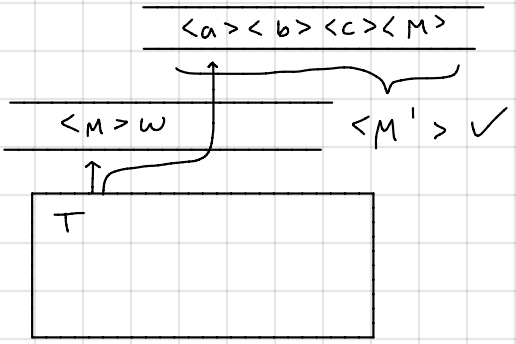
Si fijamos $\langle M \rangle w$. Podemos construir M':



$$L(M') = \begin{cases} \Sigma^* & \text{si } w \in L(M) \\ \phi & \text{si no} \end{cases}$$

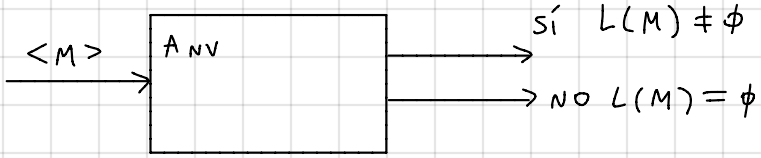


¿Cómo se puede construir T?

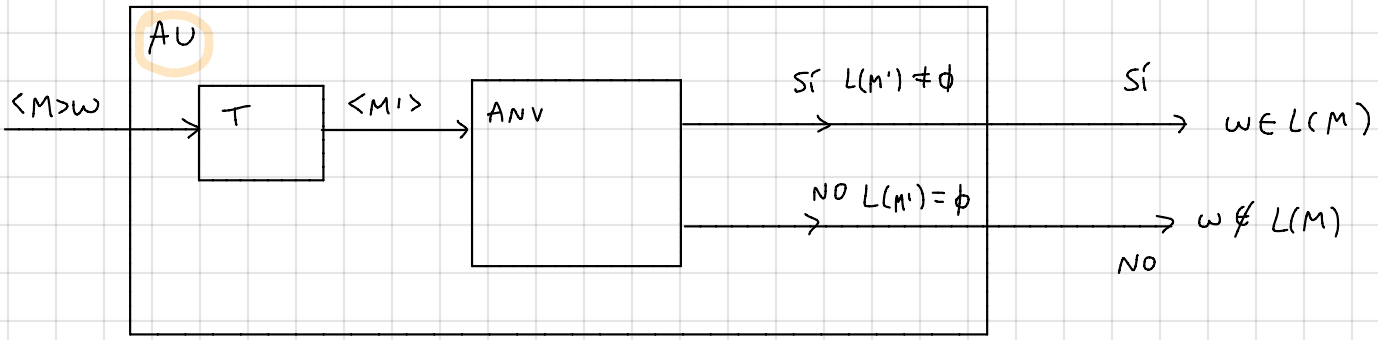


<a> : borrar input x.
<b> : transiciones para escribir w
<c> : trans. para pasar a estado inicial de M

Supongamos L_{NV} recursivo. Existe este algoritmo:



Si juntamos T y A_{NV}

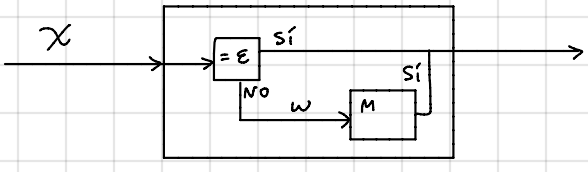


Pero ya sabemos que Au no existe! ⇒⇐

⇒ L_{NV} es rec. en. pero no recursivo.

- Por esto, L_V ($L_V = L_{NV}^C$) no puede ser rec. enum.
- $L_\varepsilon = \{ \langle M \rangle \mid L(M) = \{\varepsilon\} \}$ no es rec. enum.

Dem. Mismo truco. Fijamos <M>w y construimos M'.

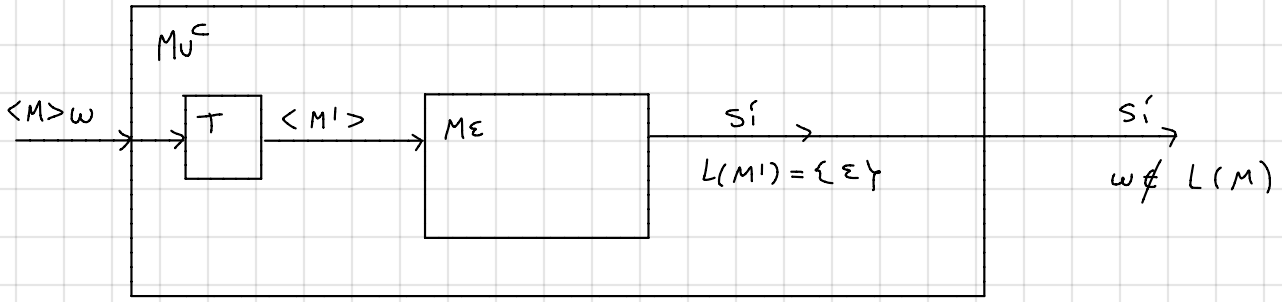


$$L(M') = \begin{cases} \{\varepsilon\} & \text{si } w \notin L(M) \\ \Sigma^* & \text{si no} \end{cases}$$

Tenemos misma T.

Supongamos L_ε es rec. enum.

Juntamos T y Mε:

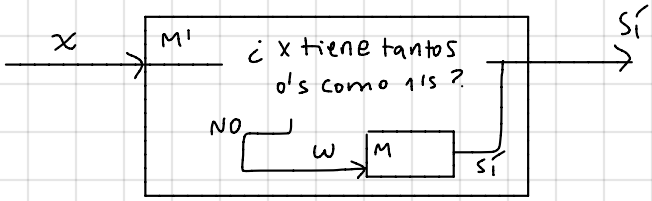


$L(M_u^c) = L_u^c$ ⇒ L_u debería ser recursivo, pero no lo es ⇒⇐

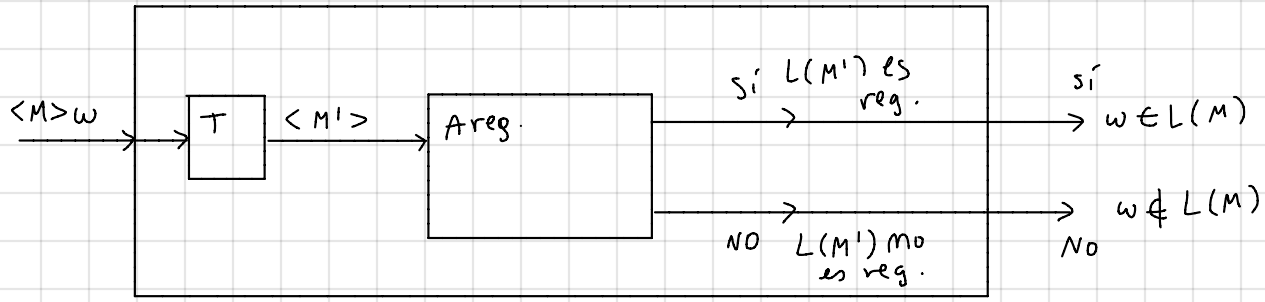
∴ L_ε no es rec. enum ⇒ L_ε es indecidible.

- $L = \{ \langle M \rangle \mid L(M) \text{ es regular} \}$, $\Sigma = \{0,1\}$, es indecidible.

$\langle M \rangle w$ fijos.



$$L(M') = \begin{cases} \{ x \mid \#_0(x) = \#_1(x) \} & \text{si } w \notin L(M) \quad \rightarrow \text{no reg.} \\ \Sigma^* & \text{si } w \in L(M) \quad \rightarrow \text{reg.} \end{cases}$$



Se concluye como antes.

- **Teorema de Rice**: Todos los problemas no triviales sobre los lenguajes de MT son indecidibles (las MT no pueden razonar sobre otras MT).
  También: todo lo que uno se pregunte sobre un programa es indecidible.
  Enunciado informal

* Ejemplos de problemas triviales:

- $L = \{ \langle M \rangle \mid L(M) \text{ es rec. enum.} \}$ siempre es cierto!
- $L = \{ \langle M \rangle \mid L(M) = L_D \}$ nunca es cierto!

- Los problemas decidibles son enumerables, el total de problemas no.
- ¿Cuánto cuesta resolver un problema? Midiendo tiempo (cant. de transiciones de una MT) y espacio (cant. de espacios en la cinta de una MT). Se pueden jerarquizar según cuánto cuestan: