La máquina de Turing y la jerarquía de la computabilidad (clases 1 y 2)

Sebastián Hernandez Corsi y Josefina Martinez Osti

Ejercicio 1. Responder breve y claramente los siguientes incisos:
1. Dados Σ = {a, b y c} y L = {$a^n b^n c^n$ | n ≥ 0}, obtener Σ*∩ L, Σ*∪ L, y L^C (es decir, el complemento de L con respecto a Σ*).

- Σ*= {λ, a, b, c, aa, ab, ac, …}
- Σ*∩ L=L
- Σ*∪ L=Σ*
- L^C=Σ*-L o lo que es lo mismo: L^c= {w∈Σ* | w∉L}

2. Definir el problema de la satisfacibilidad de las fórmulas booleanas en la forma de problema de búsqueda (visión de MT calculadora), de decisión (visión de MT reconocedora), y de enumeración (visión de MT generadora).

- MT calculadora: en este enfoque, la MT buscaría una combinación de valores de verdad que satisfaga la fórmula y la devolvería si es que existe.
- MT reconocedora: una máquina de turing de este estilo tomaría una cadena que representa una fórmula booleana de entrada y llegaría a qa o qr según si la fórmula es satisfacible o no.
- MT generadora: esta MT generaría infinitamente fórmulas booleanas aleatorias e iría verificando si son o no satisfactibles, dejando en la cinta de resultado aquellas que sí lo son.

3. ¿Qué postula la Tesis de Church-Turing?

La Tesis de Church-Turing postula que cualquier función que puede ser computada mediante un algoritmo puede ser calculada por una Máquina de Turing. Esto implica que las Máquinas de Turing marcan el límite de lo que es computacionalmente posible.

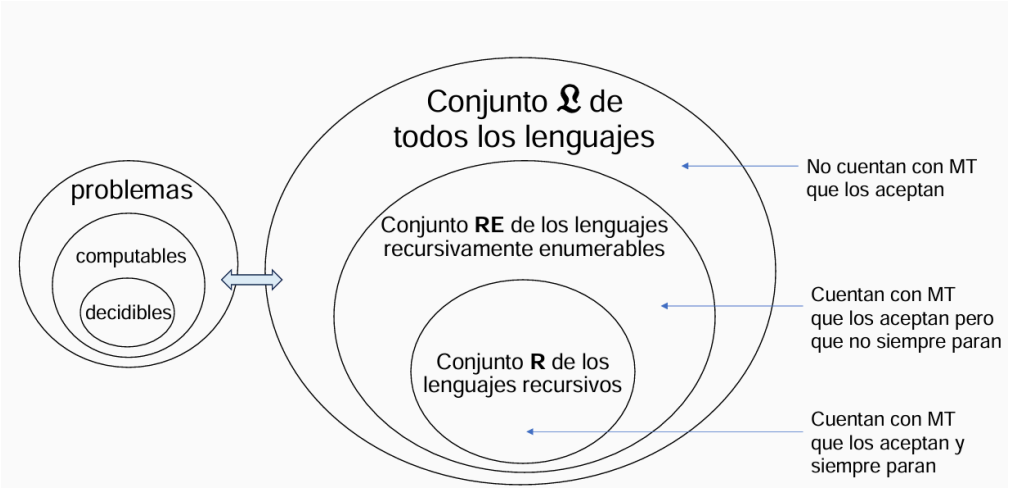
4. ¿Cuándo dos MT son equivalentes? ¿Cuándo dos modelos de MT son equivalentes?

- Máquinas equivalentes: dos MT son equivalentes cuando resuelven el mismo problema. Es decir, si para la misma entrada generan la misma salida (en caso de mt de decisión o reconocedoras) y en el caso de las generadoras sí generan las mismas salidas.
- Modelos equivalentes: dos modelos son equivalentes si tienen el mismo poder computacional. Esto significa que cualquier problema que pueda ser resuelto por un modelo debe ser también resoluble por el otro (sin importar si es más lento y complicado)

5. ¿En qué se diferencian los lenguajes recursivos, los lenguajes recursivamente enumerables no recursivos, y los lenguajes no recursivamente enumerables?

- Lenguaje recursivo (R): lenguajes para los cuales existe una MT que lo acepta y para SIEMPRE (lo puede decidir). Siempre para en qa o en qr

- Lenguaje recursivamente enumerable (RE): si existe una MT que lo acepta pero a veces loopea.
- Lenguaje no recursivamente enumerable: Si no existe una MT que ni siquiera puede aceptarlo.



6. Probar que R ⊆ RE ⊆ Ω. Ayuda: usar las definiciones.

R ⊆ RE-> lo podemos probar por definición. en R están los lenguajes con MT que los aceptan y paran siempre y en RE los lenguajes que tienen MT que los aceptan. En otras palabras, R es un subconjunto de RE

RE ⊆ Ω.-> se prueba también por definición. en Ω están todos los lenguajes con alfabeto Σ, y RE es el conjunto de lenguajes con MT que los aceptan, por lo que RE es subconjunto de Ω

7. Explicar por qué
(a) el lenguaje Σ* de todas las cadenas,
(b) el lenguaje vacío ∅, y
(c) cualquier lenguaje finito, son recursivos. Alcanza con dar la idea general.
Ayuda para (c): por cada cadena del lenguaje podría definirse como un conjunto específico de transiciones.

a- El lenguaje de **Σ*** está en **R** ya que se puede construir una **MT** que acepte al lenguaje y para siempre. Esta MT lo que haría es leer el primer caracter e inmediatamente frenar en q_a.

b- El lenguaje de ∅ está en **R** porque se puede construir una **MT** que lo acepte y pare siempre. Esta MT simplemente rechaza siempre, sin importar cuál sea la **w** que ingrese.

c- Cualquier lenguaje finito será recursivo porque podemos crear una **MT** que acepte las cadenas de **L** y rechacen las que no estén en **L**, siempre terminando por el hecho de ser finitos. Lo que haría esta **MT** sería comparar la entrada **w** con todas las cadenas de **L** y alcanzar el estado q_a si coincide con alguna cadena y en q_r en caso contrario

8. Explicar por qué no es correcta la siguiente prueba de que si L ∈ RE, también L^C ∈ RE: dada una MT M que acepta L, entonces la MT M', igual que M pero con los estados finales permutados, acepta L^C.

Para que L∈ RE tiene que haber una MT que:

- si la cadena w está en L, la máquina la acepte
- Si la cadena no está en L, la máquina o loopea o rechaza.

Para que L^c∈ RE, tiene que haber una MT que reconozca las cadenas que NO están en L.
El problema que hay al intentar probar que si L ∈ RE, también L^C ∈ RE es que si la máquina original M se quedaba loopeando, la máquina M' podría loopear en lugar de aceptar, en otras palabras, si M no terminaba nunca para algunas cadenas fuera de L, entonces M' no puede aceptar bien L^c porque algunas cadenas de la misma harán que nunca pare.
En un ejemplo concreto:
Digamos que L es el lenguaje de todas las cadenas que tienen al menos un 1. Entonces una MT que reconoce L tendría que aceptar si encuentra un 1 y si la cadena tiene solo 0 puede quedarse en un bucle infinito y no dar respuesta o rechazarla explícitamente.
L^c sería en nuestro ejemplo entonces un lenguaje de todas las cadenas que NO tienen ningún 1. Para que L^c esté en RE, tendría que tener una MT' que acepte todas las cadenas sin 1 y looper rechazar las que tiene 1.
El problema entonces es que si construimos esta MT' solo invirtiendo los estados q_a y q_r la máquina no funcionará bien ya que podría quedar loopeando cuando debería aceptar.

Ejercicio 2. Explicar (dar la idea general) cómo una MT que en un paso no puede simultáneamente modificar un símbolo y moverse, puede simular (ejecutar) una MT que sí lo puede hacer.

Para simular una mt que puede modificar un símbolo y moverse en simultáneo con una que no puede hacerlo, podemos agregar un estado intermedio extra y lograr el mismo resultado.

- Paso 1: escribir el símbolo (sin moverse) y pasar al estado intermedio
- Paso 2: moverse en la dirección que corresponda.

Ejemplo: si la MT original (la que puede hacer las dos cosas en simultáneo) tiene la transición
$(q_0,1)\rightarrow(q_1,0,R)$
En la MT_1 (la que o se mueve o escribe) podemos simularlo haciendo algo como:
$(q_0,1)\rightarrow(q_{der},0,S)$
$(q_{der},0)\rightarrow(q_1,0,R)$

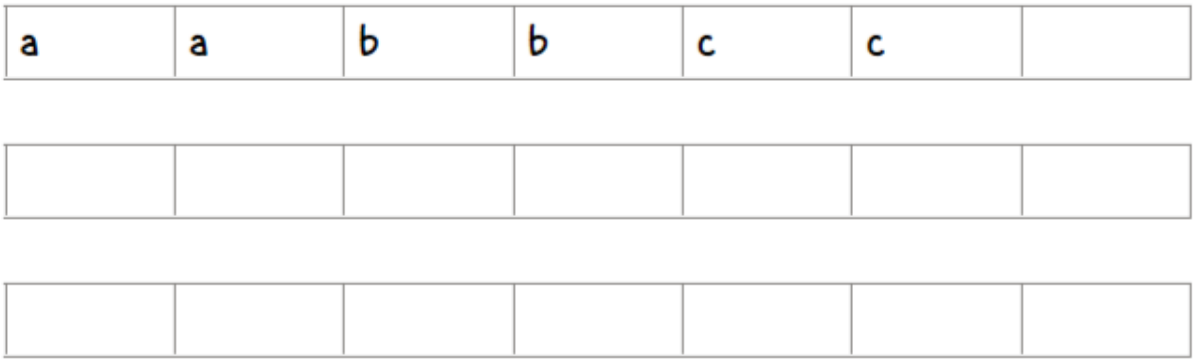
Ejercicio 3. Describir la idea general de una MT con varias cintas que acepte, de la manera más eficiente posible (menor cantidad de pasos), el lenguaje L = $\{a^n b^n c^n \mid n \geq 0\}$.

Una solución podría ser usar 3 cintas y: leer la entrada original y copiar las "a" a la cinta 2 y las "b" a la cinta 3 para luego verificar que la cantidad de "c" de la entrada original coincide con la cantidad de "a" y la de "b".
Con un poco más de detalle habría que:

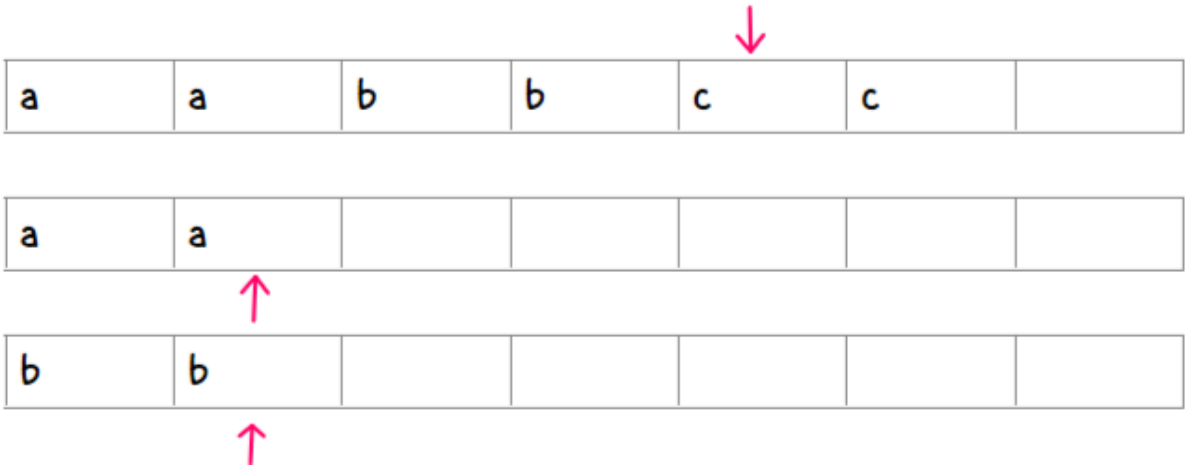
- Leer las "a" en la primera cinta. mientras haya "a" copiarla en la segunda cinta y moverse a la derecha. Si aparece una "c", un blanco o cualquier caracter antes de leer una "b", se rechaza.

- Cuando se lee la primera “b”, se comienza a copiarlas en la 3ra cinta. Si aparece un blanco o algún caracter distinto de “c”, se rechaza
- En cuanto aparece la primera c, hay que moverse en simultáneo en las 3 cintas (en la primera hacia la derecha y en las otras dos hacia la izquierda) y corroborar que las 3 tengan la misma cantidad de caracteres. Si las cintas se terminan al mismo tiempo, se acepta. Si sobra algún caracter en alguna cinta se rechaza.

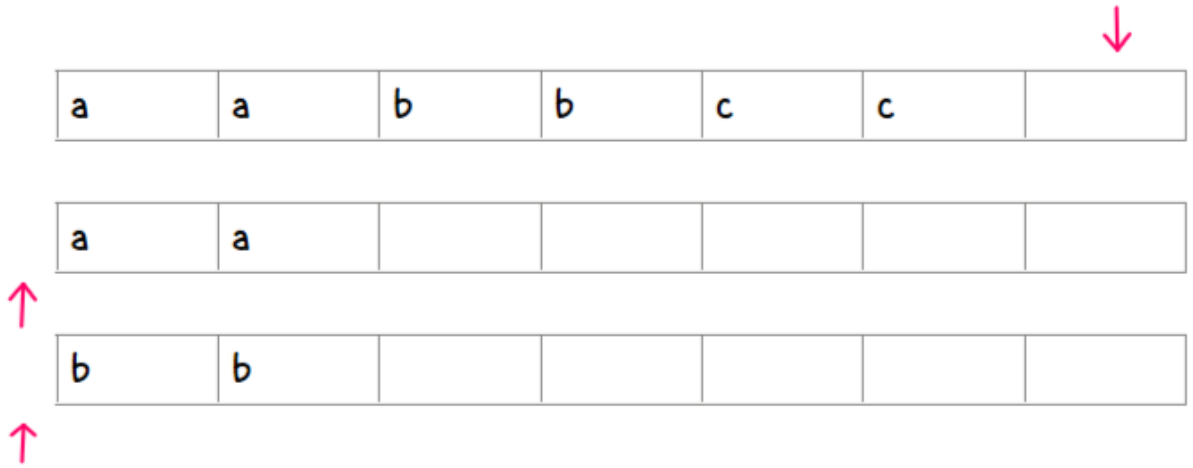
Ejemplo de una entrada



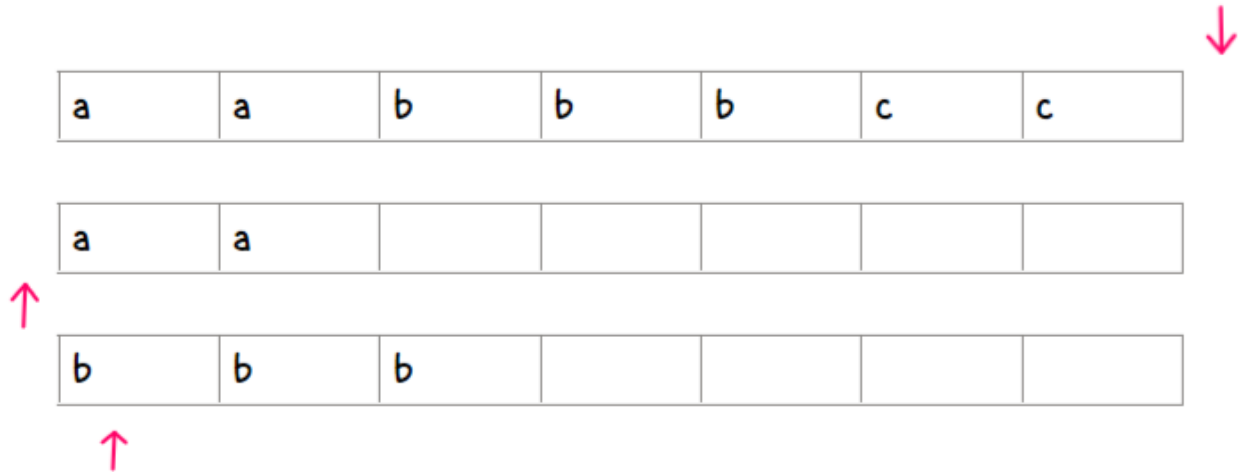
Estado en el que se copiaron las “a” y las “b” y se encontró la primera “c”



Se mueve el cabezal de la primera cinta hacia la derecha y los de las otras dos a la izquierda. Si se terminan de recorrer las 3 cadenas en simultáneo, se acepta. En caso contrario se rechaza.

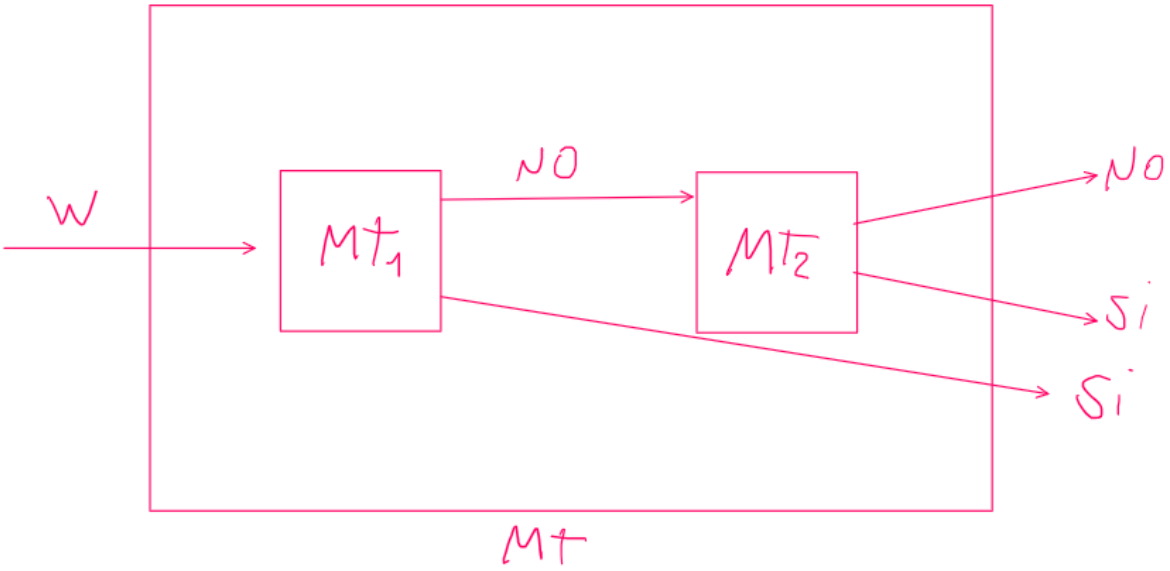


En este caso luego de 2 movimientos se acepta.



En este otro ejemplo se rechaza porque se terminan de recorrer las “a” y las “c” pero no las “b”.
Consideramos que la solución es bastante buena porque únicamente hay que realizar tantos pasos como caracteres tiene la cadena.

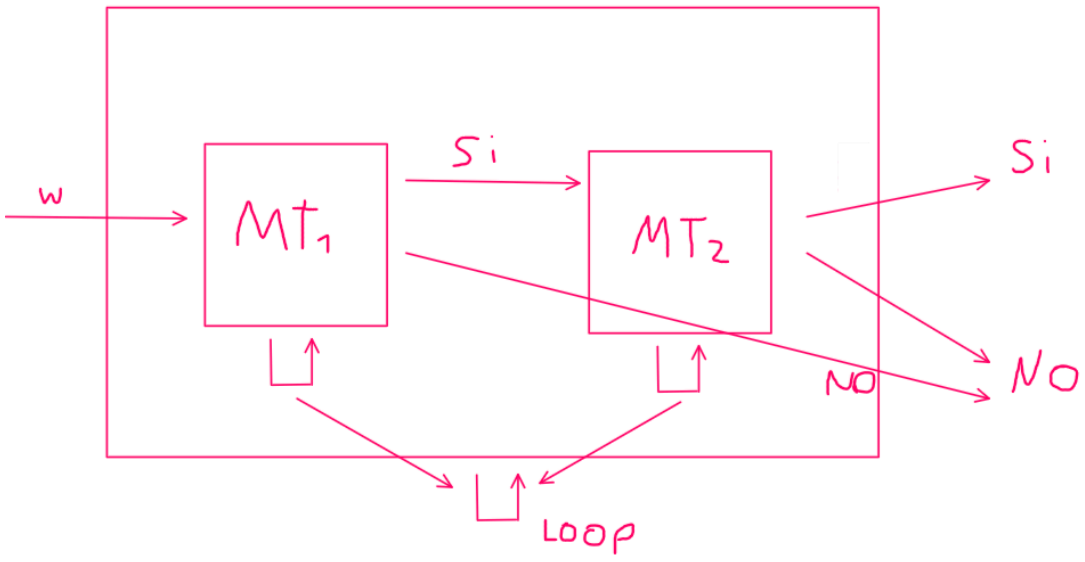
Ejercicio 4. Probar:
1. La clase R es cerrada con respecto a la operación de unión.
Ayuda: la prueba es similar a la desarrollada para la intersección.
Hay que probar que dados dos lenguajes L_1 y $L_2 \in R$ entonces $L_1 \cup L_2 \in R$. Es decir que dada una MT_1 que decide L_1 y una MT_2 que decide L_2 , también existe una MT que decide $L_1 \cup L_2$.
Podemos construir la MT de manera tal que ejecute secuencialmente MT_1 y una MT_2 y hacer que si MT_1 acepta, se acepte la cadena, pero si MT_2 rechaza, antes de rechazar la cadena se pase por MT_2 y se termine de decidir en base a esa MT.
Graficamente:



2. La clase RE es cerrada con respecto a la operación de intersección.
Ayuda: la prueba es similar a la desarrollada para la clase R.

Hay que probar que dados dos lenguajes L_1 y $L_2 \in$ RE entonces $L_1 \cap L_2 \in$ RE. Es decir que dada una MT_1 que decide L_1 y una MT_2 que decide L_2 , también existe una MT que decide $L_1 \cap L_2$.

Para poder conseguir esto y evitar falsos positivos, se debe ejecutar secuencialmente cada máquina de turing, esto se debe a que por definición de intersección, si una máquina rechaza dada una entrada w, esta entrada no puede ser aceptada por la máquina de turing universal, en cambio si la MT_1 acepta y la MT_2 acepta, la MTU aceptará, para cualquier otro caso será tomado como loop o como rechazo.
En otras palabras, para que esta MTU acepte es condición necesaria que tanto MT_1 y MT_2 acepten, si una de las dos llega a q_r MTU dará también este resultado y en caso de que una loopee no habría problema ya que la máquina general loopeara llevando a un resultado esperado dentro de lo que es el conjunto RE.
Gráficamente:



Ejercicio 5. Sean L_1 y L_2 dos lenguajes recursivamente enumerables de números naturales codificados en unario (por ejemplo, el número 5 se representa con 11111). Probar que también es recursivamente enumerable el lenguaje L = {x | x es un número natural codificado en unario, y existen y, z, tales que y + z = x, con $y \in L_1$, $z \in L_2$}. Ayuda: la prueba es similar a la vista en clase, de la clausura de la clase RE con respecto a la operación de concatenación.

Tenemos que $L_1 \in RE$ y $L_2 \in RE$. Queremos probar $L=\{x \mid y \in L_1, z \in L_2$ tal que $x=y+z\}$ también pertenece a RE.

X= Y+Z en unario no es más que concatenar Y y Z.

Ejemplo:

y=3: y=111

z=2: z=11

Entonces x=y+z=111+11=11111

Lo que sabemos con esto es que si una cadena x pertenece a L, entonces la podemos dividir en dos partes: una parte y que pertenece a L_1 y una parte z que pertenece a L_2.

Lo que debemos hacer entonces es ir tomando la cadena x que ingresa a L y probar todas las formas de dividirla en dos partes para probar que estas pertenezcan tanto a L_1 como a L_2.

Siguiendo con el ejemplo anterior, si x=11111 habría que ir probando

"" y "11111"

"1" y "1111"

"11" y "111"

"111" y "11"

"1111" y "1"

"11111" y ""

y en cada “corte” ir probando si las cadenas pertenecen a L_1 y L_2. Si se encuentra una combinación en que la primera parte pertenece a L_1 y la segunda a L_2, entonces la cadena se debe aceptar.

El problema es que como L_1 y L_2 pueden loopear, se podría dar el caso de que alguna de las dos mt quede loopeando. Por eso, ejecutar primero una y luego la otra no nos sirve.

Lo que habría que hacer es ejecutarlas “por pasos”, pero a su vez habría que ejecutar “en simultáneo” todas las posibles particiones.

Es decir que habría que ejecutar: Paso 1 de mt_1 para la 1ra partición. Paso 1 de mt_2 para la 1ra partición. Paso 1 de mt_1 para la 2da partición, paso 1 de mt_2 para la 2da partición y así hasta terminar con todas las particiones y una vez terminado eso recién saltar al paso 2.

Esa sería la forma de que si alguna de las máquinas loopea para alguna de las cadenas, nos permita seguir buscando si hay una partición que termina en q_a.

Ejercicio 6. Dada una MT M1 con alfabeto Γ = {0, 1}:

1. Construir una MT M2, utilizando la MT M1, que acepte, cualquiera sea su cadena de entrada, sii la MT M1 acepta al menos una cadena.

Ayuda para la parte (1): Si M1 acepta al menos una cadena, entonces existe al menos una cadena de símbolos 0 y 1, de tamaño n, tal que M1 la acepta en k pasos. Teniendo en cuenta esto, pensar cómo M2 podría simular M1 considerando todas las cadenas de símbolos 0 y 1 hasta encontrar eventualmente una que la acepte M1 (¡cuidándose de los casos en que M1 entre en loop!).

Tenemos que construir una MT_2 que acepte la entrada w (cualquiera sea esta) si MT_1 acepta por lo menos una palabra. Es decir, necesitamos que MT_2 simule a MT_1 sobre todas las cadenas posibles con el alfabeto dado.
La ejecución de la MT_2 sería algo como:

1. ignorar la entrada w
2. Crear una lista infinita de todas las cadenas con el alfabeto dado (“”, “0”,”1”, “00”,”01”,...)
3. Para cada cadena, ejecutar en simultáneo (“paso a paso”) una simulación de MT_1 sobre cada cadena. Es decir: paso 1 para la cadena 1, paso 1 de la cadena 2… paso 1 de la cadena n. Luego arranca con el paso 2 para cada una y así sucesivamente. Si alguna de las simulaciones llega a q_a, MT_2 acepta.

2. ¿Se puede construir además una MT M3, utilizando la MT M1, que acepte, cualquiera sea su cadena de entrada, sii la MT M1 acepta a lo sumo una cadena? Justificar.

Nos están preguntando si se puede construir una MT_3 que acepte si M_1 acepta a lo sumo 1 cadena. El problema es que lo que la MT_3 está intentando resolver es un problema no decidible.
Esto se debe a que para aceptar si MT_1 acepta a lo sumo 1 cadena habría que verificar todas las cadenas para saber que no acepta más de una, lo que es imposible de hacer en tiempo finito. En otras palabras, a pesar de que MT_3 encuentre emulando MT_1 una cadena que esta última acepta, deberá quedarse loopeando simulando MT_1 porque no sabe si MT_1 no acepta alguna otra de las infinitas cadenas que puede tener. Lo que ocurre es que nunca podríamos estar seguros de que no existe una segunda cadena que también sea aceptada. Por lo tanto, MT_3 loopearía incluso en los casos en que debería aceptar.