Trabajo Práctico Nro 2 - La jerarquía de la Computabilidad

Ejercicio 1

Responder breve y claramente los siguientes incisos:

1. ¿En qué se diferencian los lenguajes recursivos, los lenguajes recursivamente numerables no recursivos y los lenguajes no recursivamente numerables?

Lenguajes

- **Lenguajes Recursivos (R):** Son lenguajes para los cuales existe una Máquina de Turing (MT) que los *acepta y siempre se detiene*. Esto significa que para cualquier cadena de entrada w, la MT determinará si w pertenece o no al lenguaje y siempre dará una respuesta (aceptando o rechazando).
- **Lenguajes Recursivamente Enumerables no Recursivos (RE - R):** Son lenguajes para los cuales existe una MT que los acepta. Sin embargo, esta MT *no siempre se detiene* para las cadenas que no pertenecen al lenguaje. Si una cadena w está en el lenguaje, la MT la aceptará y se detendrá. Pero si w no está en el lenguaje, *la MT podría rechazar y detenerse o entrar en un bucle infinito.*
- **Lenguajes No Recursivamente Enumerables:** Estos lenguajes son aquellos para los cuales *no existe una MT que los acepte*.

En resumen, la principal diferencia radica en la existencia de una Máquina de Turing que los acepte y si esa máquina siempre se detiene para cualquier entrada. Los lenguajes recursivos tienen MTs que siempre se detienen, los recursivamente enumerables (no recursivos) tienen MTs que aceptan pero no siempre se detienen, y los no recursivamente enumerables no tienen MTs que los acepten.

2. Probar que $R \subseteq RE \subseteq \mathfrak{L}$. *Ayuda: usar las definiciones.*

Definiciones a tener en cuenta

- **L:** Un lenguaje con alfabeto Σ es un conjunto de cadenas finitas de símbolos de Σ.
- **$\mathfrak{L}$:** Es el conjunto universal de todos los lenguajes con alfabeto Σ: conjunto de todos los subconjuntos de Σ^*.
- Un lenguaje L es **recursivo** ($L \in R$) si existe una MT M que lo acepta y siempre para (lo decide).
- Un lenguaje L es **recursivamente enumerable** ($L \in RE$) si existe una MT M que lo acepta.

R ⊆ RE: Si un lenguaje L es recursivo, es decir, (L ∈ R), entonces por definición existe una MT M que lo acepta y siempre se detiene. Esta misma MT M también cumple con la definición de un lenguaje recursivamente enumerable (existe una MT M que lo acepta y se detiene en ese caso). La condición adicional para los lenguajes recursivos es que la MT M también debe detenerse para las entradas que no están en el lenguaje, lo cual no contradice la definición de un lenguaje recursivamente enumerable. Por lo tanto, *todo lenguaje recursivo es también un lenguaje recursivamente enumerable.*

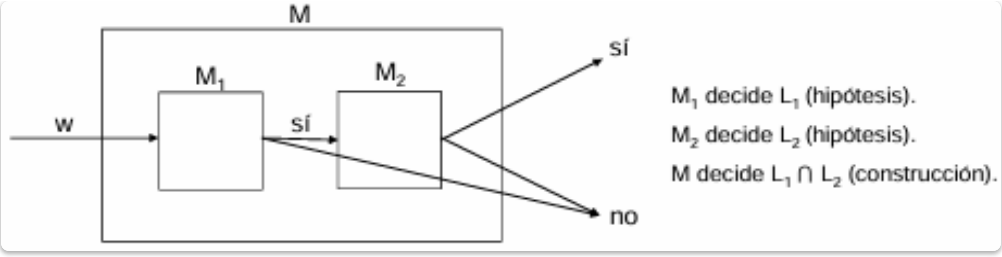
RE ⊆ $\mathfrak{L}$: Si un lenguaje L es recursivamente enumerable (L ∈ RE), entonces por definición, existe una MT M que lo reconoce. El conjunto de cadenas aceptadas por una MT define un lenguaje. Dado que todo lenguaje recursivamente enumerable es un conjunto de cadenas sobre algún alfabeto (el lenguaje aceptado por una MT), **todo lenguaje recursivamente enumerable es también un lenguaje, y por lo tanto pertenece al conjunto de todos los lenguajes $\mathfrak{L}$.

3. Dijimos en clase que el hecho de que si L es recursivo entonces L^C también lo es, significa en términos de problemas que si un problema es decidible entonces también lo es el problema contrario. ¿Qué significa en términos de problemas que la intersección de dos lenguajes recursivos es también un lenguaje recursivo?

En términos de problemas, que la intersección de dos lenguajes recursivos ($L_1 \cap L_2$) sea también un lenguaje recursivo significa que si tenemos dos problemas que son decidibles individualmente, entonces el problema que consiste en determinar si una instancia dada es una instancia positiva para ambos problemas, también es decidible.

Más específicamente:

- Si L_1 es un lenguaje recursivo, representa un problema P_1 que es decidible. Esto significa que existe una Máquina de Turing M_1 que siempre se detiene y determina si una instancia dada pertenece o no a las instancias positivas de P_1.
- Si L_2 es un lenguaje recursivo, representa un problema P_2 que es decidible. Esto significa que existe una Máquina de Turing M_2 que siempre se detiene y determina si una instancia dada pertenece o no a las instancias positivas de P_2.
- La intersección $L_1 \cap L_2$ representa el problema $P_{1\cap2}$ cuya instancia positiva es aquella que es instancia positiva tanto para P_1 como para P_2.
- El hecho de que $L_1 \cap L_2$ sea recursivo implica que existe una Máquina de Turing M que siempre se detiene y decide si una instancia dada es positiva tanto para P_1 como para P_2. Esta Máquina de Turing M puede lograr esto ejecutando secuencialmente M_1 y M_2 sobre la misma entrada. M aceptará la entrada solo si tanto M_1 como M_2 la aceptan (lo que indica que la instancia es positiva para ambos problemas). En cualquier otro caso (si M_1 rechaza o M_2 rechaza), M rechazará. Dado que tanto M_1 como M_2 siempre se detienen, M también siempre se detendrá, demostrando que $P_{1\cap2}$ es decidible.



En resumen, la propiedad de cierre de los lenguajes recursivos bajo la operación de intersección se traduce a que si podemos decidir dos problemas por separado, también podemos decidir el problema que requiere que ambas condiciones de los problemas originales se cumplan simultáneamente.

4. 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.

Definiciones a tener en cuenta

- El complemento de un lenguaje L, denotado como L^C, contiene todas las cadenas sobre el alfabeto que no están en L.

La prueba que se propone para demostrar que si $L \in RE$, entonces $L^C \in RE$ es incorrecta debido a la naturaleza de los lenguajes recursivamente enumerables (RE) y el comportamiento de las Máquinas de Turing (MT) que los aceptan.

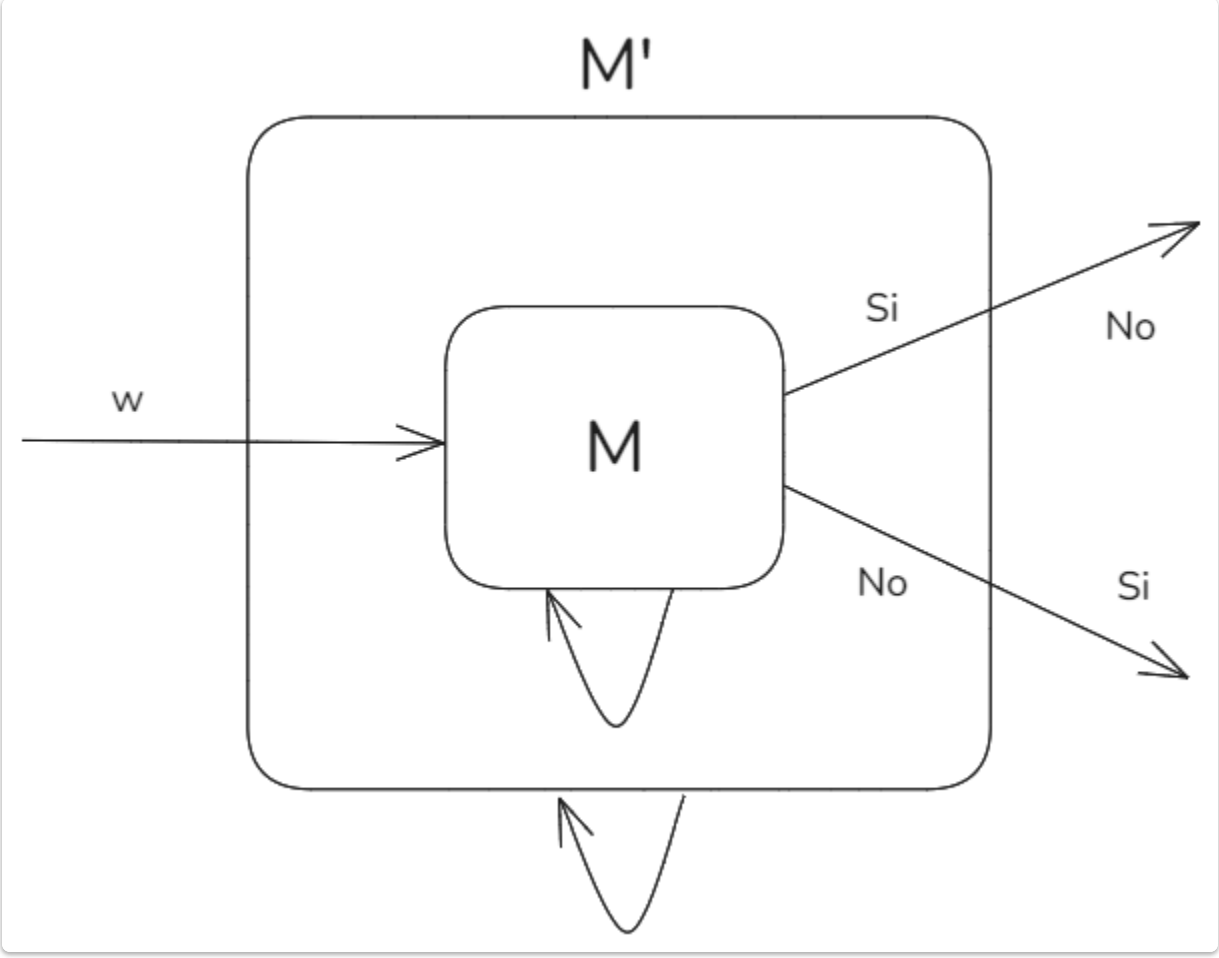
Tomamos una MT M que acepta L y creamos una nueva MT M' que es idéntica a M excepto que sus estados de aceptación (q_A) y rechazo (q_R) están permutados. La idea es que si M acepta w (porque $w \in L$), entonces M' rechazaría w. Y si M rechazaba w (porque $w \notin L$), entonces M' aceptaría w.

El problema fundamental con esta prueba radica en la posibilidad de que la MT M no se detenga para las entradas $w \notin L$.

Prueba

Consideremos un caso donde $w \notin L$. Según la definición de RE, la MT M podría:

1. Rechazar w y detenerse. En este caso, la MT M' con los estados permutados aceptaría w, lo cual es correcto ya que $w \in L^C$.
2. No detenerse (loopear). En este caso, la MT M' también no se detendría al procesar w, ya que las funciones de transición (δ) son las mismas excepto por el intercambio de los estados finales. Para que M' aceptara L^C, debería detenerse y aceptar todas las cadenas $w \notin L$. El hecho de que M' entre en un bucle infinito significa que no está aceptando w, aunque w pertenezca a L^C.



Por lo tanto, la MT M' construida de esta manera no necesariamente acepta L^C. Puede que rechace correctamente algunas cadenas que no están en L (aquellas para las que M se detiene y acepta), pero no aceptará correctamente todas las cadenas que no están en L para las cuales M no se detiene.

En resumen, la prueba es incorrecta porque no considera el caso en que la MT M para un lenguaje RE no recursivo puede no detenerse para las entradas que no pertenecen al lenguaje. La MT M' con los estados finales permutados heredaría este comportamiento de no detención, lo cual no cumple con el requisito de aceptación para el complemento del lenguaje.

5. ¿Qué lenguajes de la clase CO-RE tienen MT que los aceptan? ¿También los deciden?

En términos de lenguajes de la clase CO-RE:

- Los lenguajes que pertenecen a la clase CO-RE son aquellos cuyos complementos pertenecen a la clase RE (lenguajes recursivamente enumerables).
- Si un lenguaje L pertenece a CO-RE, entonces su complemento L^C pertenece a RE. Esto significa que existe una Máquina de Turing (MT) que acepta L^C. Para las cadenas $w \in L^C$, esta MT se detendrá y aceptará. Para las cadenas $w \notin L^C$ (es decir, $w \in L$), la MT podría detenerse y rechazar o no detenerse (loopear).

¿Qué lenguajes de la clase CO-RE tienen MT que los aceptan?

- **Los lenguajes que están en la intersección de RE y CO-RE (es decir, RE ∩ CO-RE) tienen MTs que los aceptan.** R = RE ∩ CO-RE, por lo tanto, todos los lenguajes recursivos (R) pertenecen a CO-RE y tienen MTs que los aceptan (y que siempre paran).
- **Los lenguajes que están en CO-RE pero no en R (CO-RE - R) no cuentan con MTs que los acepten.** Si un lenguaje $L \in CO - RE - R$, entonces su complemento $L^C \in RE - R$. Esto significa que existe una MT que acepta L^C pero no siempre se detiene para las cadenas que no están en L^C (es decir, las cadenas en L). Por definición de CO-RE, no se garantiza la existencia de una MT que acepte directamente L.

¿También los deciden?

- **Solo los lenguajes que pertenecen a la clase R (los lenguajes recursivos) son decididos por MTs.** Una MT decide un lenguaje si lo acepta y siempre para para cualquier entrada (aceptando si la entrada está en el lenguaje y rechazando si no lo está).
- **Los lenguajes en CO-RE - R no son decidibles.** Si un lenguaje $L \in CO - RE - R$, aunque su complemento L^C sea aceptado por una MT que no siempre para, no existe una MT que siempre pare y decida L. Si existiera una MT que decidiera L, entonces L sería recursivo, lo que contradice el hecho de que $L \in CO - RE - R$.

En resumen:

- Los lenguajes de la clase CO-RE que tienen MTs que los aceptan son precisamente los lenguajes recursivos (R), que son un subconjunto de CO-RE.
- Estos mismos lenguajes recursivos (R) son también los únicos en CO-RE que son decididos por MTs (porque la MT que los acepta siempre se detiene).
- Los lenguajes que están en CO-RE pero no son recursivos (CO-RE - R) no tienen MTs que los acepten.

6. Probar que el lenguaje Σ* de todas las cadenas y el lenguaje vacío ⌀ son recursivos. Alcanza con plantear la idea general. *Ayuda: encontrar MT que los decidan.*

Lenguaje Σ^*

- **Idea general:** Podemos construir una MT M_{Σ^*} que, al recibir cualquier cadena de entrada w, inmediatamente la acepte y se detenga.
- **Construcción conceptual de M_{Σ^*}:**
  - Al iniciar en el estado inicial q_0 con la cadena w en la cinta, la MT M_{Σ^*} podría tener una transición que la lleve directamente al estado de aceptación q_A sin realizar ninguna otra operación (o realizando operaciones triviales como mover el cabezal y volver a su posición inicial).

    - No importa cuál sea la cadena de entrada w (incluso la cadena vacía λ), la MT siempre alcanzará el estado de aceptación q_A y se detendrá.
- **Justificación:** Dado que toda cadena $w \in \Sigma^*$ pertenece al lenguaje, y nuestra MT M_{Σ^*} acepta cualquier entrada w y siempre se detiene en el estado de aceptación, por definición, Σ^* es un lenguaje recursivo.

Lenguaje vacío $\emptyset$

- **Idea general:** Podemos construir una MT $M_\emptyset$ que, al recibir cualquier cadena de entrada w, inmediatamente la rechace y se detenga.
- **Construcción conceptual de $M_\emptyset$:**
    - Al iniciar en el estado inicial q_0 con la cadena w en la cinta, la MT $M_\emptyset$ podría tener una transición que la lleve directamente al estado de rechazo q_R sin realizar ninguna otra operación (o realizando operaciones triviales).
    - No importa cuál sea la cadena de entrada w, la MT siempre alcanzará el estado de rechazo q_R y se detendrá.
- **Justificación:** Dado que ninguna cadena w pertenece al lenguaje vacío ($\forall w, w \notin \emptyset$), y nuestra MT $M_\emptyset$ rechaza cualquier entrada w y siempre se detiene en el estado de rechazo, por definición, $\emptyset$ es un lenguaje recursivo.

7. Probar que todo lenguaje finito es recursivo. Alcanza con plantear la idea general. *Ayuda: encontrar una MT que lo decida (pensar cómo definir sus transiciones para cada una de las cadenas del lenguaje).*

Para probar que todo lenguaje finito es recursivo, debemos demostrar que para cualquier lenguaje finito L, existe una Máquina de Turing (MT) que lo decide. Esto significa que la MT debe aceptar todas las cadenas que pertenecen a L, rechazar todas las cadenas que no pertenecen a L, y siempre detenerse para cualquier entrada.

Sea $L = w_1, w_2, \ldots, w_k$ un lenguaje finito, donde $w_1, w_2, \ldots, w_k$ son las k cadenas distintas que componen el lenguaje (si L es el lenguaje vacío $\emptyset$, ya demostramos que es recursivo). Podemos construir una MT M que decide L de la siguiente manera:

- **Idea general:** La MT M tomará una cadena de entrada w y la comparará secuencialmente con cada una de las cadenas $w_1, w_2, \ldots, w_k$ que definen el lenguaje finito L.
- **Construcción conceptual de la MT M:**
    - Al recibir una cadena de entrada w, la MT M comenzará a compararla con la primera cadena del lenguaje, w_1.
    - Si w es idéntica a w_1, la MT M transicionará a un estado de aceptación q_A y se detendrá.

- Si w no es idéntica a w_1, la MT M procederá a comparar w con la siguiente cadena del lenguaje, w_2.
- Este proceso de comparación continuará para cada una de las cadenas w_i en el lenguaje L, desde $i = 1$ hasta k.
- Si en algún momento durante estas comparaciones, la MT encuentra que w es idéntica a alguna w_i (para $1 \leq i \leq k$), entonces la MT aceptará w y se detendrá.
- Si después de comparar w con todas las k cadenas en L, la MT no ha encontrado ninguna coincidencia, esto significa que w no pertenece al lenguaje L. En este caso, la MT transicionará a un estado de rechazo q_R y se detendrá.

- **Justificación de que M siempre se detiene:** El proceso de comparación es finito. Hay un número finito de cadenas en L (k cadenas). Para cualquier entrada w, la MT realizará a lo sumo k comparaciones completas. Después de la última comparación (con w_k), si no hubo aceptación previa, la MT rechazará y se detendrá. Por lo tanto, para cualquier cadena de entrada, la MT M siempre realizará un número finito de pasos y se detendrá, ya sea en el estado de aceptación o en el estado de rechazo.

Ejercicio 2

Considerando la Propiedad 2 estudiada en clase:

1. ¿Cómo implementaría la copia de la entrada w en la cinta 2 de la MT M?

La copia de la entrada w de la cinta 1 a la cinta 2 se puede implementar mediante una secuencia de transiciones de la MT M de la siguiente manera:

- Inicialmente, la entrada w se encuentra en la cinta 1, y ambas cabezas de lectura/escritura están posicionadas al inicio de las cintas. La cinta 2 está inicialmente en blanco.
- La MT M entraría en un estado dedicado a la operación de copia.
- Mientras la cabeza de la cinta 1 lea un símbolo s (que no sea el símbolo blanco que indica el fin de la entrada):
  - La MT M escribirá el mismo símbolo s en la posición actual de la cabeza de la cinta 2.
  - Ambas cabezas (la de la cinta 1 y la de la cinta 2) se moverán una posición a la derecha.
  - La MT M permanecerá en el estado de copia.

2. ¿Cómo implementaría el borrado del contenido de la cinta 2 de la MT M?

El borrado del contenido de la cinta 2 se puede implementar con otra secuencia de transiciones de la MT M:

- Después de la ejecución de M_1 sobre w en la cinta 2, la cabeza de la cinta 2 podría estar en cualquier posición. Para borrar el contenido, primero necesitamos posicionar la cabeza al inicio del contenido que se desea borrar. Esto podría implicar mover la cabeza hacia la izquierda hasta encontrar un símbolo blanco que indique el inicio de la región utilizada.
- Una vez que la cabeza de la cinta 2 esté posicionada al inicio del contenido a borrar, la MT M entraría en un estado dedicado al borrado.
- Mientras la cabeza de la cinta 2 lea un símbolo que no sea el símbolo blanco (indicando que aún hay contenido de la ejecución anterior):
  - La MT M escribirá el símbolo blanco en la posición actual de la cabeza de la cinta 2.
  - La cabeza de la cinta 2 se moverá una posición a la derecha.
  - La MT M permanecerá en el estado de borrado.

Ejercicio 3

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.*

Para probar que la clase R (lenguajes recursivos) es cerrada con respecto a la operación de unión, debemos demostrar que si L_1 y L_2 son lenguajes recursivos, entonces su unión $L_1 \cup L_2$ también es un lenguaje recursivo.

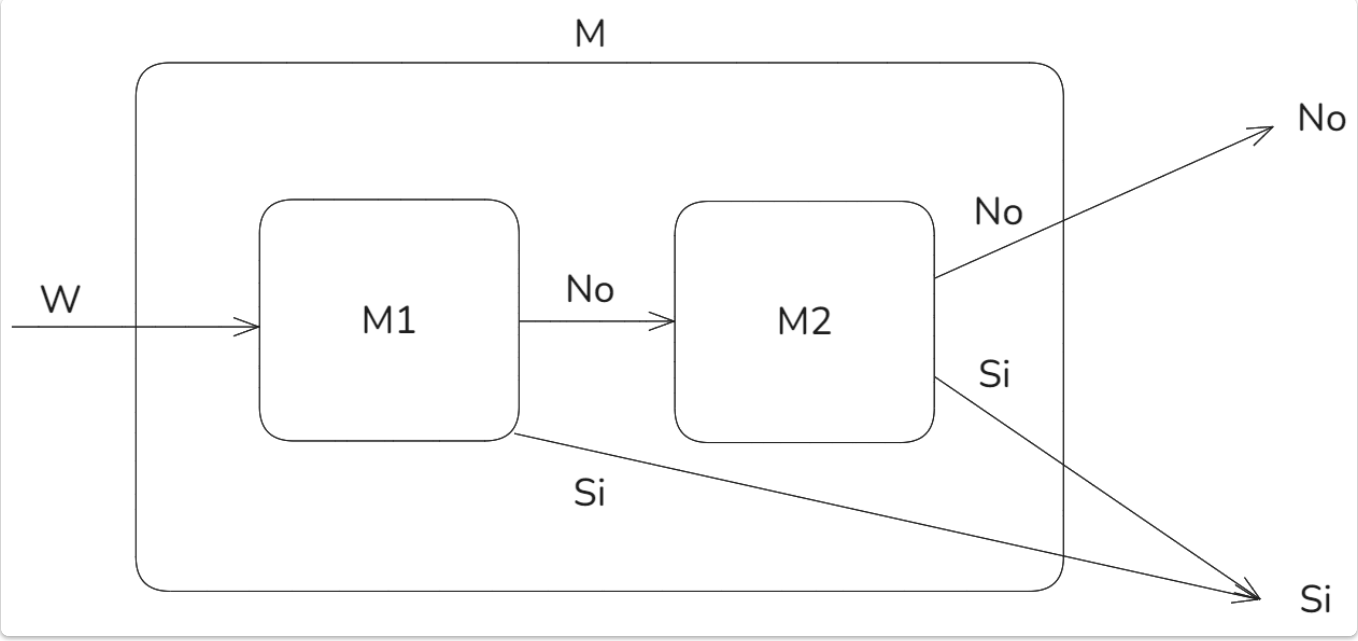
Dado que $L_1 \in R$ y $L_2 \in R$, por definición, existen dos Máquinas de Turing:

- **M_1 que decide L_1:** Para cualquier entrada w, M_1 se detiene y acepta si $w \in L_1$, y se detiene y rechaza si $w \notin L_1$.
- **M_2 que decide L_2:** Para cualquier entrada w, M_2 se detiene y acepta si $w \in L_2$, y se detiene y rechaza si $w \notin L_2$.

Nuestro objetivo es construir una MT M que decida el lenguaje $L_1 \cup L_2 = w \mid w \in L_1 \text{ o } w \in L_2$.

Idea General

La MT M tomará una cadena de entrada w y simulará la ejecución de M_1 y M_2 sobre w. Si al menos una de ellas acepta, entonces w pertenece a $L_1 \cup L_2$, y M debe aceptar. Si ambas rechazan, entonces w no pertenece a $L_1 \cup L_2$, y M debe rechazar. Debido a que tanto M_1 como M_2 siempre se detienen (ya que deciden L_1 y L_2 respectivamente), nuestra MT M también siempre se detendrá, ya sea en el estado de aceptación o de rechazo. Por lo tanto, M decide $L_1 \cup L_2$.



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.*

Para probar que la clase RE (lenguajes recursivos enumerable) es cerrada con respecto a la operación de intersección, debemos demostrar que si L_1 y L_2 son lenguajes recursivos enumerables, entonces su intersección $L_1 \cap L_2$ también es un lenguaje recursivo enumerable.

Dado que $L_1 \in RE$ y $L_2 \in RE$, por definición, existen dos Máquinas de Turing:

- M_1 **que decide** L_1: Para cualquier entrada w, M_1 se detiene y acepta si $w \in L_1$, y puede detenerse y rechazar si $w \notin L_1$ o puede no detenerse y quedarse loopenado.
- M_2 **que decide** L_2: Para cualquier entrada w, M_2 se detiene y acepta si $w \in L_2$, y puede detenerse y rechazar si $w \notin L_2$ o puede no detenerse y quedarse loopenado.

Nuestro objetivo es construir una MT M que decida el lenguaje $L_1 \cap L_2 = w \mid w \in L_1 \text{ y } w \in L_2$.

Idea General

La idea general es simular la ejecución de M1 y M2 secuencialmente sobre una misma entrada w.

Dado una entrada w, la MT M operará de la siguiente manera:

1. Ejecutar M1 sobre la entrada w.
2. Si M1 rechaza w (se detiene en su estado de rechazo), entonces M también rechaza w. Si M1 no se detiene, entonces M tampoco se detiene en esta rama de la ejecución.
3. Si M1 acepta w (se detiene en su estado de aceptación), entonces M procede a ejecutar M2 sobre la misma entrada w. Para esto, podemos imaginar que M guarda la entrada w y

reinicia o simula la ejecución de M2 con w. También se puede pensar en una MT con múltiples cintas donde una cinta contiene la entrada y las otras se usan para simular M1 y M2.

4. Si M2 acepta w (se detiene en su estado de aceptación), entonces M acepta w.
5. Si M2 rechaza w (se detiene en su estado de rechazo), entonces M rechaza w. Si M2 no se detiene, entonces M tampoco se detiene.

De esta construcción, podemos observar lo siguiente:

- M aceptará la entrada w si y solo si tanto M1 como M2 aceptan w. Esto se debe a que M solo llega al paso de ejecutar M2 si M1 ya aceptó. Si M2 también acepta, entonces M acepta.
- Si w no pertenece a L1, entonces M1 rechazará o no se detendrá, y en ambos casos, M no aceptará.
- Si w pertenece a L1 pero no a L2, entonces M1 aceptará, pero M2 rechazará o no se detendrá, y en ambos casos, M no aceptará.

Por lo tanto, M reconoce exactamente el lenguaje L1 ⋂ L2. Aunque M1 o M2 puedan no detenerse para entradas que no están en sus respectivos lenguajes, la construcción asegura que M acepta precisamente cuando ambas M1 y M2 aceptan, cumpliendo con la definición de que L1 ⋂ L2 es recursivamente enumerable.

Ejercicio 4

Sean L_1 y L_2 dos lenguajes recursivamente numerables de números naturales codificados en unario (por ejemplo, el número 5 se representa con 11111). Probar que también es recursivamente numerable el lenguaje L = {x | x es un número natural codificado en unario, y existen y, z, tales que y + z = x, con y ∈ L_1, z ∈ 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.*

Para probar que el lenguaje
$L = x \mid x \text{ es un número natural codificado en unario, y existen } y, z, \text{ tales que } y + z = x, \text{ con } y \in L_1, z \in$
es recursivamente enumerable, dado que L_1 y L_2 son lenguajes recursivamente enumerables de números naturales codificados en unario, podemos construir una Máquina de Turing (MT) M que acepte L.

Dado que $L_1 \in RE$ y $L_2 \in RE$, existen MT M_1 que acepta L_1 y M_2 que acepta L_2. Esto significa que para una entrada w en unario:

- Si w representa un número $n \in L_1$, entonces M_1 se detiene en un estado de aceptación (q_A). Si $n \notin L_1$, M_1 puede detenerse en un estado de rechazo (q_R) o no detenerse.
- Si w representa un número $m \in L_2$, entonces M_2 se detiene en un estado de aceptación (q_A). Si $m \notin L_2$, M_2 puede detenerse en un estado de rechazo (q_R) o no detenerse1 .

Queremos construir una MT M que, dada una entrada x en unario (representada por 1^x), acepte si existen números naturales y y z tales que $y + z = x$, donde la representación unaria de y está en L_1 y la representación unaria de z está en L_2.

La idea general es similar a la prueba de que la clase RE es cerrada bajo la operación de concatenación. En el caso de la concatenación, para una entrada w, se consideran todas las posibles formas de partir w en dos subcadenas v_1 y v_2, y se simulan las MT que aceptan los lenguajes correspondientes en paralelo.

En nuestro caso, la suma de números naturales en codificación unaria corresponde a la concatenación de las representaciones unarias. Si y se representa como 1^y y z como 1^z, entonces $y + z = x$ se representa como la concatenación $1^y1^z = 1^{y+z} = 1^x$.

Construimos la MT M de la siguiente manera:

1. Recibir la entrada x en unario (1^x).
2. Iterar sobre todas las posibles particiones de x en dos números no negativos y y z tales que $y + z = x$. Esto se puede lograr considerando todas las posibles formas de dividir la cadena 1^x en dos subcadenas contiguas (la primera de longitud y y la segunda de longitud z), incluyendo los casos donde $y = 0$ (cadena vacía para M_1) o $z = 0$ (cadena vacía para M_2).
3. Para cada partición (y, z), donde 1^y es la representación unaria de y (si $y = 0$, es la cadena vacía λ) y 1^z es la representación unaria de z (si $z = 0$, es λ):
   - Simular la ejecución de M_1 con la entrada 1^y.
   - Simular la ejecución de M_2 con la entrada 1^z.
   - Es crucial que estas simulaciones se realicen de forma intercalada o "en paralelo". Esto significa que M debe ejecutar un número finito de pasos de M_1 y luego un número finito de pasos de M_2, y repetir este proceso. Esto es necesario porque M_1 o M_2 podrían no detenerse si sus entradas no pertenecen a L_1 o L_2 respectivamente, y no queremos que una simulación no terminada impida que se considere otra partición o que se detecte una posible aceptación.
   - Si en algún momento, la simulación de M_1 acepta 1^y Y la simulación de M_2 acepta 1^z, entonces M acepta la entrada x (1^x) y se detiene.
4. Si se han considerado todas las posibles particiones de x en y y z, y en ninguna de ellas tanto M_1 aceptó 1^y como M_2 aceptó 1^z, entonces M no acepta x. M podría detenerse en un estado de rechazo o no detenerse.

Si existe un par (y, z) tal que $y + z = x$, con $y \in L_1$ y $z \in L_2$, entonces M_1 aceptará la representación unaria de y y M_2 aceptará la representación unaria de z. Debido a la simulación paralela (o intercalada) de M_1 y M_2 para cada partición, M eventualmente encontrará esta partición y aceptará x.

Si no existe tal par (y, z), entonces para todas las particiones, al menos una de las máquinas (M_1 o M_2) no aceptará su entrada. Por lo tanto, M nunca llegará a la condición de aceptación en el paso 3.

Dado que hemos construido una MT M que acepta exactamente el lenguaje L, y M acepta una cadena si y solo si pertenece a L (aunque puede no detenerse si la cadena no pertenece a L), concluimos que L es recursivamente enumerable.

Ejercicio 5

Dada una MT M_1 con alfabeto Γ = {0, 1}:

1. Construir una MT M_2, utilizando la MT M_1, que acepte, cualquiera sea su cadena de entrada, si la MT M_1 acepta al menos una cadena.

La MT M_2 funcionará de manera independiente de su propia entrada. Su objetivo es determinar si M_1 acepta alguna cadena en el conjunto de todas las cadenas posibles sobre el alfabeto {0, 1}. Para lograr esto, M_2 simulará la ejecución de M_1 en todas las posibles cadenas de entrada (sobre {0, 1}) de forma sistemática.

La construcción de M_2 se puede describir de la siguiente manera:

M_2 ignorará su propia cadena de entrada. Internamente, M_2 procederá a realizar la siguiente secuencia de acciones:

1. Iterar sobre la longitud de las posibles cadenas de entrada para M_1. Comenzaremos considerando cadenas de longitud 0, luego longitud 1, longitud 2, y así sucesivamente. Sea n la longitud actual considerada, comenzando con $n = 0$.
2. Generar todas las cadenas binarias de longitud n. Para cada longitud n, M_2 generará todas las posibles cadenas compuestas por los símbolos 0 y 1. Por ejemplo, para $n = 0$, la cadena es "λ" (cadena vacía); para $n = 1$, las cadenas son "0" y "1"; para $n = 2$, son "00", "01", "10", "11", y así sucesivamente.
3. Simular M_1 en cada cadena generada. Para cada cadena w de longitud n generada en el paso anterior, M_2 simulará la ejecución de M_1 con w como entrada.
4. Limitar el número de pasos de la simulación. Para evitar que M_2 entre en un bucle infinito si M_1 no se detiene en una entrada particular, debemos limitar el número de pasos de la

simulación.

- Una estrategia es limitar el número de pasos de la simulación para una cadena de longitud n a un valor que dependa de n pero el número de pasos que M_1 necesita para aceptar una cadena no está directamente ligado a la longitud de esa cadena. M_1 podría aceptar una cadena corta después de muchos pasos, o una cadena larga en pocos pasos.
- Una estrategia más clara es combinar la longitud de la cadena y el número de pasos utilizando un límite combinado que se incrementa a medida que exploramos más cadenas.
  - Introducimos un límite s que representa tanto la longitud máxima de las cadenas de entrada para M_1 que M_2 considera en una etapa, como el número máximo de pasos que M_2 simula de M_1 para cada una de esas cadenas.
  - Comenzamos con un valor de $s = 0$ (o quizás $s = 1$ para incluir cadenas no vacías dependiendo del problema).
  - Para cada valor actual de s, M_2 realiza las siguientes acciones:
    1. Genera todas las posibles cadenas w sobre el alfabeto de entrada de M_1 cuya longitud $|w|$ es menor o igual a s.
    2. Para cada una de estas cadenas w, M_2 simula la ejecución de M_1 en w durante un máximo de s pasos.
    3. Si durante cualquiera de estas simulaciones, M_1 alcanza su estado de aceptación, entonces M_2 acepta.
  - Después de simular M_1 para todas las cadenas de longitud $\leq s$ durante a lo sumo s pasos, M_2 incrementa el valor de s (por ejemplo, a $s + 1$) y repite el proceso.

> **Nota**
>
> La clave de esta estrategia combinada es que el límite s crece independientemente de la longitud de cualquier cadena particular que M_1 pueda aceptar. Si M_1 acepta una cadena w_0 de longitud n_0 en k_0 pasos, eventualmente el valor de s será lo suficientemente grande como para que $s \geq n_0$ y $s \geq k_0$. En ese punto (o antes, cuando s cumpla estas condiciones), M_2 generará w_0 y la simulará durante suficientes pasos (s) para observar la aceptación de M_1, lo que permitirá que M_2 complete su tarea.

5. Continuar la búsqueda si no se encuentra una cadena aceptada por M_1. Si la simulación de M_1 en todas las cadenas de longitud hasta s no resulta en una aceptación dentro de los s pasos, M_2 continúa con el siguiente valor de s.

Conclusiones

- Si M_1 acepta al menos una cadena w_0 de longitud n_0 en k_0 pasos, entonces en la iteración de M_2 donde el límite combinado s sea mayor o igual a $\max(n_0, k_0)$, M_2 eventualmente generará la cadena w_0 y simulará M_1 en ella durante al menos k_0 pasos. Durante esta simulación, M_1 alcanzará su estado de aceptación, y por lo tanto, M_2 aceptará.
- Si M_1 no acepta ninguna cadena, entonces para cualquier cadena w y cualquier número finito de pasos de simulación, M_1 nunca alcanzará su estado de aceptación. En este caso, M_2 continuará simulando indefinidamente sin nunca alcanzar su propio estado de aceptación (lo cual es consistente con el requerimiento de que M_2 acepte si M_1 acepta al menos una cadena).

2. ¿Se puede construir además una MT M_3, utilizando la MT M_1, que acepte, cualquiera sea su cadena de entrada, si la MT M_1 acepta a lo sumo una cadena? Justificar.

Determinar si una Máquina de Turing arbitraria acepta a lo sumo una cadena es una pregunta sobre una propiedad semántica del lenguaje que reconoce. Basándonos en la comprensión de la computabilidad y la existencia de problemas indecidibles como el Problema de la Parada, se considera que propiedades no triviales sobre los lenguajes reconocidos por Máquinas de Turing son, en general, indecidibles. Por lo tanto, no es posible construir una MT M_3 que siempre decida si M_1 acepta a lo sumo una cadena.

Es importante notar que esto no significa que para algunas MT M_1 específicas podamos determinar si aceptan a lo sumo una cadena.

Ayuda para la parte (1): si M_1 acepta al menos una cadena, entonces existe al menos una cadena de símbolos 0 y 1, de tamaño n, tal que M_1 la acepta en k pasos. Teniendo en cuenta esto, pensar cómo M_2 podría simular M_1 considerando todas las cadenas de símbolos 0 y 1 hasta encontrar eventualmente una que M_1 acepte (¡cuidándose de los casos en que M_1 entre en loop!).