Práctico nro 2 FTC

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?

Se diferencian en que máquinas de turing o no los aceptan y su comportamiento.

Lenguajes Recursivos (R):

- Un lenguaje L es **recursivo** (pertenece al conjunto R) si existe una Máquina de Turing M que lo **acepta y siempre para**.
- En otras palabras, una MT que decide un lenguaje recursivo siempre da una respuesta ("sí" o "no") para cualquier entrada posible. Por esta razón, los problemas correspondientes a lenguajes recursivos se denominan **problemas decidibles**.
- Una propiedad importante es que si un lenguaje L es recursivo, entonces su **complemento** L^C también es recursivo. Esto se debe a que podemos construir una nueva MT que invierte los estados de aceptación y rechazo de la MT original.
- Además, la clase R es **cerrada** bajo las operaciones de unión ∪ e intersección ∩, y también bajo la operación de concatenación.

Lenguajes Recursivamente Enumerables No Recursivos (RE - R):

- Un lenguaje L es **recursivamente enumerable** (pertenece al conjunto RE) si existe una Máquina de Turing M que lo **acepta**. Esto significa que para cualquier cadena de entrada w:
    - Si w pertenece a L, entonces M eventualmente se detiene en su estado de aceptación qA.
    - Si w no pertenece a L, entonces M puede detenerse en su estado de rechazo qR o puede **entrar en un bucle infinito** (no detenerse).
- Los problemas correspondientes a lenguajes recursivamente enumerables se denominan **problemas computables no decidibles**. Sabemos que existen lenguajes que son recursivamente enumerables pero no recursivos.
- Un ejemplo clásico de un lenguaje en RE - R es el **problema de la parada (Halting Problem, HP)**. Existe una MT que puede aceptar todas las codificaciones de MTs que se detienen en una entrada dada, pero no existe

una MT que pueda decidir para todas las posibles entradas si una MT dada se detendrá o no.
- Si un lenguaje L pertenece a RE - R, entonces su complemento L^C **no pertenece a RE** (y por lo tanto tampoco a R), sino que pertenece al conjunto CO-RE menos R (CO-RE - R). El conjunto CO-RE contiene los complementos de los lenguajes en RE. Si tanto un lenguaje L como su complemento L^C fueran recursivamente enumerables, entonces L sería recursivo.

Lenguajes No Recursivamente Enumerables (𝓡 - RE):

- Un lenguaje L es **no recursivamente enumerable** si **no existe ninguna Máquina de Turing que lo acepte**. Esto significa que no podemos construir un algoritmo (representado por una MT) que pueda reconocer todas las cadenas que pertenecen al lenguaje.
- Estos lenguajes están fuera del alcance de la computabilidad tal como la definen las Máquinas de Turing.
- Ejemplos de lenguajes no recursivamente enumerables incluyen el lenguaje de los **enunciados aritméticos verdaderos** y el lenguaje de los conjuntos finitos de figuras poligonales que pueden cubrir el plano (problema de la teselación del plano). Para estos problemas, ni el lenguaje mismo ni su complemento son recursivamente enumerables, lo que los sitúa fuera de los conjuntos RE y CO-RE.

En resumen, la jerarquía de la computabilidad distingue los lenguajes por la capacidad de las MT para aceptarlos y detenerse:

- **Lenguajes Recursivos (R):** Existe una MT que los acepta y siempre se detiene (decide). Tanto el lenguaje como su complemento son decidibles.
- **Lenguajes Recursivamente Enumerables No Recursivos (RE - R):** Existe una MT que los acepta, pero puede no detenerse para las cadenas que no pertenecen al lenguaje. Su complemento no es recursivamente enumerable.
- **Lenguajes No Recursivamente Enumerables (𝓡 - RE):** No existe ninguna MT que los acepte.

2. Probar que R ⊆ RE ⊆ 𝓡.

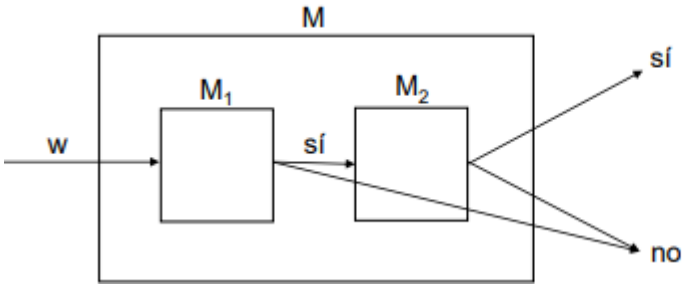
Demostración:

1. R⊆RE:Por definición, un lenguaje en R, es decir, existe un algoritmo que siempre termina y decide correctamente si una instancia es "sí" o "no". Si un problema está en R, entonces también está en RE, porque RE incluye todos los lenguajes para los cuales existe un algoritmo que puede identificar instancias "sí" (aunque no necesariamente las instancias "no").
2. Por lo tanto, R⊆RE

3. RE⊆ℒ: La clase RE contiene todos los lenguajes para los cuales existe un algoritmo que puede identificar instancias "sí". L (todos los lenguajes) incluye todos los lenguajes formales, independientemente de si son decidibles o no.

3. Dijimos en clase que el hecho de que si L es recursivo entonces LC 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?

La propiedad de que la intersección de dos lenguajes recursivos es también recursiva significa que la combinación de dos problemas decidibles mediante una condición "y" (intersección) resulta en otro problema decidible. Esto refuerza la idea de que la clase de problemas decidibles es "cerrada" bajo intersección, lo cual es una propiedad útil en la teoría de la computación.



M_1 decide L_1 (hipótesis).
M_2 decide L_2 (hipótesis).
M decide $L_1 \cap L_2$ (construcción).

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

Esta prueba es incorrecta porque supone incorrectamente que L terminara, es decir asume que es recursiva, esto lo asume porque las máquinas de lenguajes RE se pueden quedar colgadas en estados negativos lo que haría que nunca terminará de ejecutarse M' aunque la respuesta sea sí.

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

Solo aquellos de CO-RE que estan en RE también, o sea RE.

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

Ambos son recursivos porque la MT en el primer caso no debe hacer nada, entra una cadena w y la acepta, porque cualquiera cadena w están sigma estrella.

La segunda máquina siempre rechaza puesto que dado una cadena w, ninguna cadena está en el conjunto aceptado así que rechaza.

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)

Todo lenguaje finito es recursivo porque se puede crear una MT que compare la entrada con todas las cadenas del lenguaje y responda.
Al ser finito siempre termina con una cantidad finita de pasos, rechace o acepte.

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?

- Mientras el símbolo en la cinta 1 no sea blanco:
    - Leer el símbolo s en la cinta 1.
    - Escribir s en la cinta 2.
    - Mover la cabeza de la cinta 1 a la derecha.
    - Mover la cabeza de la cinta 2 a la derecha.

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

- Mientras el símbolo en la cinta 2 no sea blanco:
    - Escribir blanco (B) en la cinta 2.
    - Mover la cabeza de la cinta 2 a la derecha.

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.

Dado un lenguaje L1 y un lenguaje L2 decidibles y sus maquinas de turing M1 y M2, nosotros construiremos una maquina de turing M3 que hara lo siguiente:
Dada una cadena w, simula M1

- Si da positivo termina aceptando.
- Si falla resetea y vuelve al inicio para ejecutar M2 sobre la cadena original.

Simula M2

- Si da positivo acepta
- Si falla rechaza.

Esta máquina M3 nunca se queda loopeando ya que M1 y M2 no lo hacen, y acepta la cadena.

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.

A ver, 2 lenguajes L1 y L2 RE, con sus respectivas máquinas de turing M1 y M2, dado una máquina de turing M3 que hace lo mismo que el caso anterior:

- Simula w en cada máquina.
- Si las 2 dan positivo entonces acepta
- Si una queda loopeando o falla entonces se rechaza

No tiene sentido hacerlo concurrente (creo) puesto que requiere que todas terminen en qA para aceptar, si una ya loopea o falla es rechazo aunque no llegue a ejecutarse la otra.

Ejercicio 4. Sean L1 y L2 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 ∈ L1, z ∈ L2}. Ayuda: la prueba es similar a la vista en clase, de la clausura de la clase RE con respecto a la operación de concatenación.

Lógica: Esencialmente necesitamos una máquina M que puede tomar la entrada w, de ella aplicarle la máquinas M1 y M2, que acepten en L1 y L2 respectivamente, a cada una de sus posibles combinaciones, de forma concurrente (alternada).
La suma de números unarios es como si nosotros uniéramos 2 cadenas de unos, es como concatenar, siguiendo el ejemplo de la concatenación:

- Fraccionamos la cadena, dado que la cadena es de tamaño k, la dividimos en k-p (siendo p el paso actual), desde 0 hasta k es la cadena para la M1 y de k para adelante es para la M2.
- Simulamos de forma alternada los pasos de estas máquinas, si una falla debemos aumentar el paso actual y volver al paso de fraccionar la cadena.
- Si ambas aceptan entonces se prueba y termina.

Ejercicio 5. 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.

Mientras la M1 no acepte una cadena:

1. Género cadenas con longitud p, que inicia en 0,
2. Pruebo M1 sobre la cadenas generadas, durante máximo p pasos
3. Cuando terminó de generar cadenas de longitud p, aumentó p
4. Vuelvo paso 1 con p aumentado

La máquina M2 entonces prueba en orden canónico todas las cadenas posibles, corta cuando debe terminar, y acepta cuando M1 lo hace o queda loopeando hasta que encuentre 1.

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.

La diferencia radica en solo una cadena debe ser aceptada, entonces para mi NO se puede hacer una MT M3 que lo haga, porque para el caso positivo debe loopear sobre todas las cadenas, infinitas.

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 M1 acepte (¡cuidándose de los casos en que M1 entre en loop!).