Práctica 1

Agustina Sol Rojas y Antonio Felix Glorioso Ceretti

Ejercicio 1.

1. ¿Qué es un problema computacional de decisión? ¿Es el tipo de problema más general que se puede formular?

   Un problema computacional de decisión es aquel que dependiendo de ciertas condiciones predefinidas deberá responder si o no de acuerdo a la entrada que se reciba. No, el tipo de problema más general que se puede formular son los problemas de búsqueda, donde la respuesta puede ser cualquier valor.

2. Dados $\Sigma = \{a, b, c\}$ y $L = \{a^n b^n c^n \mid n \geq 0\}$, obtener $\Sigma^* \cap L$, $\Sigma^* \cup L$, y el complemento de L con respecto a Σ^*.

   $\Sigma^* = \{ \lambda, a, b, c, aa, ...\}$

   $\Sigma^* \cap L = L$

   $\Sigma^* \cup L = \Sigma^*$

   el complemento de L con respecto a Σ^* es $\Sigma^* - L$

3. En la clase teórica 1 se hace referencia al problema de satisfactibilidad de las fórmulas booleanas. Formular las tres formas del problema, teniendo en cuenta las tres visiones de MT consideradas: calculadora, aceptadora o reconocedora, y generadora.

   Calculadora: Dada una fórmula booleana, devuelve los valores que la hacen satisfactible.
   Aceptadora: Dada una fórmula booleana devuelve si es satisfactible o no.
   Generadora: Genera todas las fórmulas booleanas satisfactibles.

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

   Postula que todo aquello que es computable puede ser calculado por una Máquina de Turing o cualquier otro modelo de cómputo equivalente. Es importante mencionar que no se puede construir físicamente o teóricamente ninguna máquina más potente que una MT.

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

Dos MT son equivalentes si ambas aceptan el mismo lenguaje. Dos modelos de MT son equivalentes si dada una MT de un modelo existe una MT equivalente del otro.

6. ¿En qué difieren entre sí los lenguajes recursivos, los lenguajes recursivamente numerables no recursivos, y los lenguajes no recursivamente numerables?

   Un lenguaje es recursivo si y solo si existe una Maquina de Turing **ML** que lo acepta y siempre se detiene.
   Un lenguaje es recursivamente numerables no recursivo si y solo si existe una Maquina de Turing **ML** que lo acepta o en su defecto loopea sobre un input (nunca se detiene).

   Un lenguaje es no recursivamente numerable si y solo si no existe una Maquina de Turing **ML** que lo acepte.

7. Probar que $R \subseteq RE \subseteq \mathfrak{L}$.

   $R \subseteq RE$ → se puede probar por definición ya que en R están los lenguajes que tienen MT que los aceptan y siempre se detienen y en RE los lenguajes que tienen MT que los aceptan. Por lo tanto R es un subconjunto de RE.

   $RE \subseteq \mathfrak{L}$ → se puede probar por definición ya que $\mathfrak{L}$ es el conjunto de **todos** los lenguajes definidos sobre el alfabeto Σ y RE es el conjunto de los lenguajes que tienen MT que los aceptan, por lo tanto RE es un subconjunto de $\mathfrak{L}$.

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

   Los únicos lenguajes de la clase CO-RE que tienen MT que los aceptan son los lenguajes que pertenecen a R. Todo lenguaje que este en CO-RE – RE no posee una Máquina de Turing que lo acepta, si no que en realidad existe una MT que acepta a su complemento (y no necesariamente para).

9. Justificar por qué los lenguajes universal Σ* y vacío ∅ son recursivos.

   Σ* es recursivo ya que existe una Maquina de Turing que lo acepta y siempre se detiene, esta es aquella que luego de leer el primer símbolo de cualquier cadena acepta. Esta MT siempre se detiene.
   ∅ es recursivo ya que existe una Maquina de Turing que lo acepta y siempre se detiene, esta es aquella que luego de leer el primer símbolo de cualquier cadena rechaza, haciendo que el lenguaje de la maquina sea vacío. Esta MT siempre se detiene.

10. Justificar por qué un lenguaje finito es recursivo.

Un lenguaje finito es recursivo porque se puede construir una Máquina de Turing que lo acepte si la entrada coincide con alguno de los elementos del lenguaje o la rechace en el caso contrario.

11. Justificar por qué si L1 ∈ CO-RE y L2 ∈ CO-RE, entonces (L1 ⋂ L2) ∈ CO-RE.

    Si L1 ∈ CO-RE existe un $L1^c$ que ∈ RE. Si L2 ∈ CO-RE existe un $L2^c$ que ∈ RE.

    $L1^c \cup L2^c \in RE$ por lema 2

    Aplicando la Ley De Morgan:

    $L1^c \cup L2^c \in RE = (L1 \cap L2)^c \in RE$

    Como $(L1 \cap L2)^c \in RE$ entonces (L1 ⋂ L2) ∈ CO-RE (por definición)

Ejercicio 2.

Construir una MT, con cualquier cantidad de cintas, que acepte de la manera más eficiente posible el lenguaje $L = \{a^n b^n c^n \mid n \geq 0\}$.

Comentario: Plantear primero la idea general.

1. Idea general

   Utilizamos una MT con 2 cintas.

   - Copia las “a” de la primera cinta en la segunda cinta.
   - Luego recorre ambas cintas (hacia la izquierda en la segunda cinta y hacia la derecha en la primera cinta) comparando las “a” con las “b”.
   - Si la cantidad de “a” es igual a la de “b” ahora compara las “a” de la segunda cinta con las “c” de la primera cinta.

2. Construcción de la MT:

   La MT M = { Q , Σ , Γ , δ, q0 , qA , qR }
   Q = { q0, q1, q2, qA, qR }
   Σ = { a, b, c }
   Γ = { a, b, c, B }

   q0: copiar “a” en la segunda cinta.
   q1: comparar la cantidad de “a” con las “b” de la primera cinta.
   q2: comparar la cantidad de “a” con las “c” de la primera cinta.

	a, B	a, a	a, b	a, c	b, B	b, a	b, b	b, c	c, B	c,a	c,b	c,c	B,B
q0	q0 a, R a, R				q1 b, S B, L								
q1						q1 b, R a, L			q2 c, S B, R				
q2										q2 c, R a, R			qA B, S B, S

Ejercicio 3

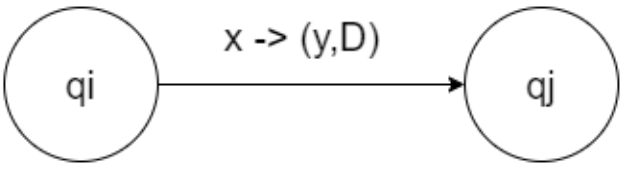
Explicar (informal pero claramente) cómo simular una MT por otra que en un paso no pueda simultáneamente modificar un símbolo y moverse.

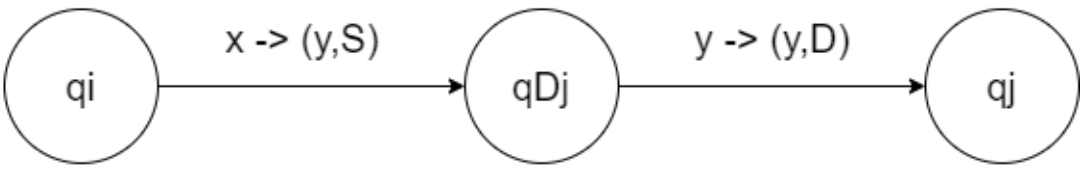
M = <Q, Σ, Γ, δ, q0, qA, qR>

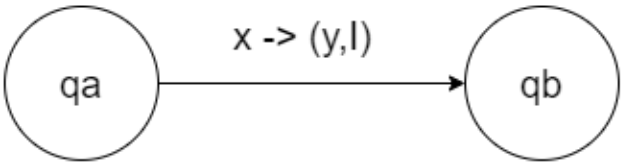
M' = <Q', Σ, Γ, δ', q0, qA, qR>

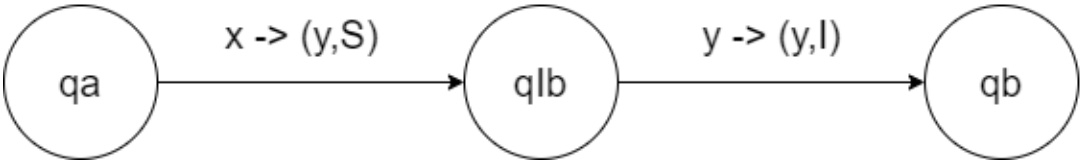
δ': Q' x Γ → Q' ∪ {qA, qR} x Γ x {D, I,S}

- Ambos empiezan en el estado q0. x, y ∈ Γ. x =/ y. qDj, qIb ∈ Q'. (estados que recuerdan cosas)
- Se agregan las siguientes transiciones (solo reemplazan a las transiciones de M que rompen con la regla de M')
- Cuando se quiere cambiar el símbolo y mover el cabezal:









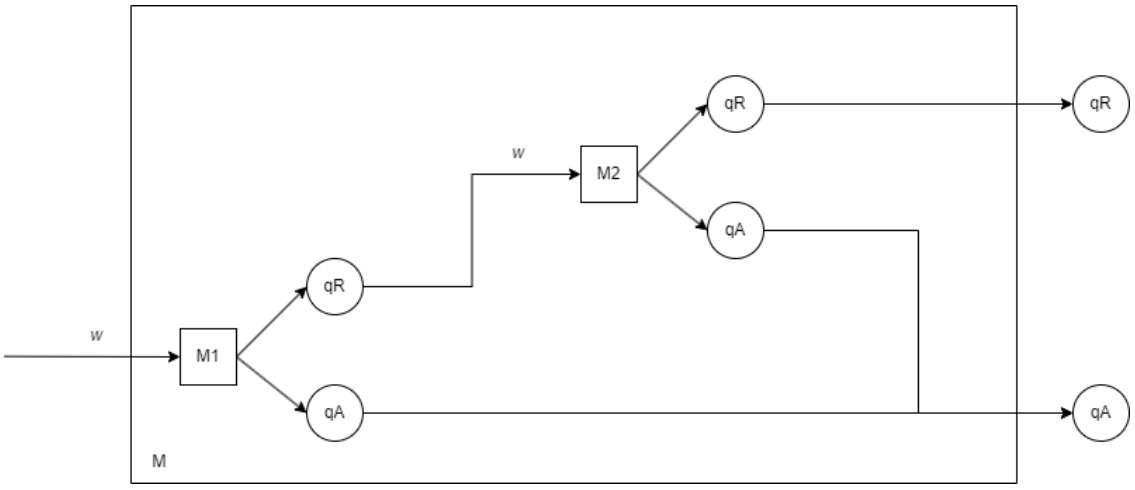
Básicamente hay un estado intermedio que depende de la dirección a donde me quiero mover y del estado al cual voy a pasar (para poder recordar a donde debo ir). Como el símbolo lo cambio antes de pasar a ese estado intermedio, no hace falta recordarlo. Va a haber tantos estados intermedios como direcciones y estados debo recordar.

Ejercicio 4

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

    1. <u>Idea general</u>

    Dadas dos MT M1 y M2 que respectivamente aceptan L1 y L2 y paran siempre, la idea es construir una MT M que acepte L1 ∪ L2 y pare siempre.



    2. <u>Construcción</u>

- M tiene 2 cintas.

- Dada la entrada w en la cinta 1, M hace:

    1. Copia w en la cinta 2.
    2. Ejecuta M1 sobre w en la cinta 2. Si M1 para en qA , entonces M para en qA. Si M1 para en qR, entonces:
        - Borra el contenido de la cinta 2 y copia de nuevo w en la cinta 2.
        - Ejecuta M2 sobre w en la cinta 2. Si M2 para en qR , entonces M para en qR. Si M2 para en qA, entonces M para en qA.

3. <u>Prueba de correctitud de la construcción.</u>

    L = L1 ⋃ L2

    M reconoce solo las entradas que son reconocidas por M1 o M2, es decir, reconoce las entradas que son reconocidas por alguna de las dos máquinas (la unión). Si se trata de una entrada que es aceptada por ambas maquinas o que es rechazada por una maquina pero no por la otra, M la va a aceptar. Si se trata de una entrada rechazada por ambas maquinas, M la va a rechazar. Por lo tanto L = L1 ⋃ L2

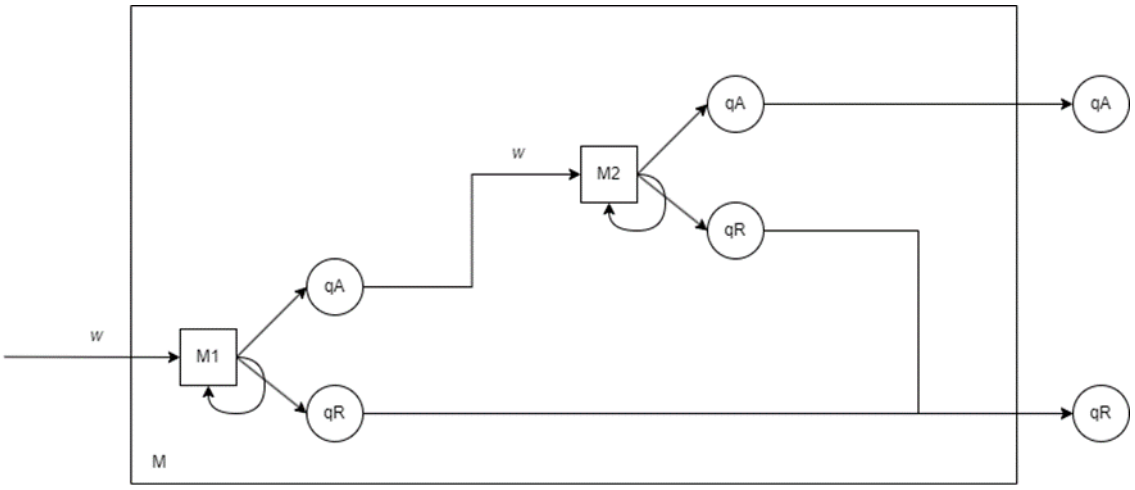
    L ∈ R

    Como M1 o M2 siempre se van a detener, M al ser una MT que simula la ejecución de las mismas y hace un copiado o borrado finito, siempre se va a detener.

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.

    1. <u>Idea general</u>

        Dadas dos MT M1 y M2 que respectivamente aceptan L1 y L2, la idea es construir una MT M que acepte L1 ⋂ L2 (L = L1 ⋂ L2).



2. Construcción

- M tiene 2 cintas.

- Dada la entrada w en la cinta 1, M hace:

    1. Copia w en la cinta 2.
    2. Ejecuta M1 sobre w en la cinta 2. Si M1 para en qR , entonces M para en qR. Si M1 loopea entonces M loopeara (rechaza). Si M1 para en qA, entonces:
        - Borra el contenido de la cinta 2 y copia de nuevo w en la cinta 2.
        - Ejecuta M2 sobre w en la cinta 2. Si M2 para en qR , entonces M para en qR. Si M2 loopea entonces M loopeara (rechaza). Si M2 para en qA, entonces M para en qA

3. Prueba de correctitud de la construcción.

L = L1 ⋂ L2

M reconoce solo las entradas que son reconocidas por M1 y M2, es decir, reconoce las entradas que son reconocidas por ambas maquinas (la intersección). Si se trata de una entrada que es rechazada por una maquina pero no por la otra, M no la va a aceptar. Si se trata de una entrada rechazada por ambas maquinas, M la va a rechazar. Por lo tanto L = L1 ⋂ L2

L ∈ RE

Como L = L(M), existe una máquina que lo acepta. Si M1 o M2 se quedan loopeando M se va a quedar loopeando.

Ejercicio 5

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}.

1. Idea general

   El lenguaje L1 • L2 contiene todas las cadenas w = x1x2 , tales que la subcadena x1 L1 y la subcadena x2 L2 .
   Sea M1 una MT que acepta el lenguaje L1 y M2 una MT que acepta el lenguaje L2 porque dichos lenguajes pertenecen a RE.
   Hay que construir una MT M que acepte el lenguaje L1 • L2.

   Dado un input w con n símbolos, M hace:

   - M ejecuta M1 a partir de los primeros 0 símbolos de w, y M2 a partir de los últimos n símbolos de w. Si en ambos casos se acepta, entonces M acepta.
   - Si no, M hace lo mismo que en (1) pero ahora con el 1er símbolo y los últimos (n – 1) símbolos de w. Si en ambos casos se acepta, entonces M acepta.
   - Si no, M hace lo mismo que en (1) pero ahora con los primeros 2 y los últimos (n – 2) símbolos de w. Si en ambos casos se acepta, entonces M acepta.

     Y así siguiendo, con 3 y (n – 3), 4 y (n – 4), ..., hasta llegar a n y 0 símbolos de w. Si en ninguno de los casos se acepta, entonces M rechaza.

   Como existe la posibilidad de loops por parte de M1 y M2, M debe ejecutarlas “en paralelo”:
   M primero debe hacer ejecuciones de 1 paso de M1 y M2 con todas las posibles particiones de w, luego ejecuciones de 2 pasos con todas las particiones, luego ejecuciones de 3 pasos con todas las particiones, y así siguiendo hasta eventualmente aceptar.

2. Construcción

   - M tiene 3 cintas.
   - En la cinta 1 tiene la entrada w.
   - En las cintas 2 y 3 ejecuta M1 y M2, respectivamente.
   - El K inicial es 1
   - La MT M hace:

1. Copia w de la cinta 1 a las cintas 2 y 3 realizando la división del input a partir de los primeros 0 símbolos de w, y M2 a partir de los últimos n símbolos de w.
2. Ejecuta k pasos de M1 en la cinta 2.
3. Ejecuta k pasos de M2 en la cinta 3.
4. Si M1 y/o M2 rechazan, dependiendo del estado de la maquina hay dos opciones:
    - Si todavía no se hicieron todas las posibles particiones del input, se continúa dividiendo el input, se copia en las cintas 2 y 3, y se vuelve al punto 2 con el mismo k.
    - Si se hicieron todas las particiones del input:
        - Si solo una maquina rechazo, se incrementa k (la cantidad de pasos) y se vuelve al punto 1 partiendo desde la división inicial de input.
        - Si ambas maquinas rechazaron entonces M rechaza.
5. Si se ejecutaron los k pasos de ambas maquinas y ninguna rechazo o acepto se incrementa k y se vuelve al punto 1.

- M acepta si ambas maquinas aceptan.

- En cada iteración memoriza los estados y posiciones de las 2 ejecuciones

3. <u>Prueba de correctitud de la construcción.</u>

    L = L1 • L2

    M reconoce solo las entradas donde ambas subcadenas son reconocidas por M1 y M2.
    - Si w pertenece a L1 • L2 entonces w puede ser divida en dos partes x1 y x2, donde x1 pertenece a L1 y x2 pertenece a L2. Por lo tanto M1 aceptara a x1 y M2 aceptara a x2, lo que llevara a M a aceptar w
    - Si w no pertenece a L1 • L2, por construcción M1 o M2 eventualmente rechazaran, lo que llevara a M a rechazar w.

    L ∈ RE

    Si alguna maquina en algún punto rechazan loopeando, M va a loopear. Si ambas maquinas aceptan, M va a aceptar.

Ejercicio 6

Dada una MT M1 con alfabeto Σ = {0, 1}:

1. Construir una MT M2 que determine si L(M1) tiene al menos una cadena.
2. ¿Se puede construir además una MT M3 para determinar si L(M1) tiene a lo sumo una cadena? Justificar.

1. M2 va a generar todos los inputs en forma de pares (i, j) en orden de su suma, i+j, y entre los de igual suma en orden creciente de i (i es la cadena y j son los pasos por ejecutarse sobre esa cadena). Ej: (1, 1); (1, 2); (2, 1); (1, 3); (2, 2); (3, 1), ...
   Por cada par (i, j) generado se simulan j pasos de la MT M1 sobre el wi generado. Si M1 acepta la cadena i en esos j pasos, M2 acepta.
   Lo que va a suceder es que, luego de que se ejecute la maquina con la cadena i en j pasos, se va a pasar a la siguiente en el orden de la suma, la maquina nunca se va a quedar loopeando sobre el mismo input.

2. No se puede construir una MT M3 porque en la situación en la que se debería aceptar (L(M1) tiene una sola cadena) nunca se terminaría de hacer ya que se va a seguir buscando una segunda cadena y la maquina “loopeara”, rechazando dicho lenguaje a pesar que deberia aceptarlo ya que el mismo tiene una sola cadena. La MT M3 no tiene forma de saber si el lenguaje realmente tiene una sola cadena, ya que cuando va a buscar una segunda lo hara de forma infinita, ya que la cantidad de strings es infinita, haciendo que la máquina “loopee”.

   Ejemplo: L(M1) = {1010}

   La MT M3 va a recibir la <M1> y va a ir generando las cadenas sobre las cuales ejecutara <M1>. Cuando M3 llegue a la cadena 1010 va a incrementar un contador k (ya que M1 la aceptara) que lleva la cuenta de las cadenas del lenguaje y va a seguir buscando por otra cadena que nunca encontrara (ya que M1 siempre va a rechazar) y “loopeara” cuando en realidad debería aceptar.