Trabajo Práctico Nro 3 - Indecibilidad

Ejercicio 1

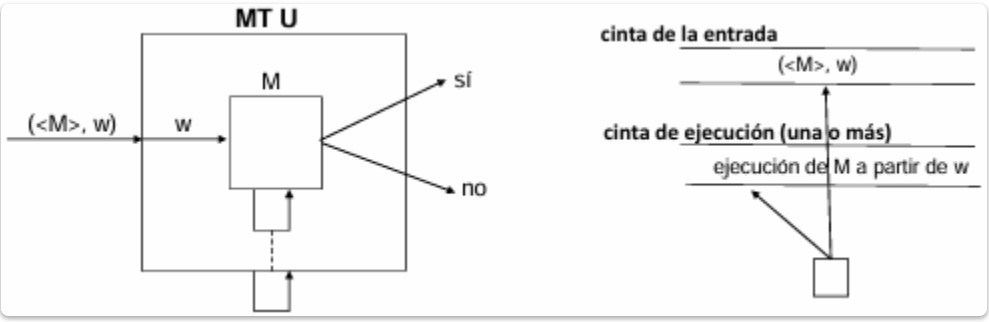
¿Qué es una MT universal?

Una **Máquina de Turing Universal (MT U)** es una máquina de Turing teórica capaz de simular cualquier otra máquina de Turing. Esto significa que puede ejecutar cualquier algoritmo computable que pueda ser implementado por una máquina de Turing específica.

Aspectos clave de una MT universal:

- **Entrada:** Una MT universal recibe como entrada la *descripción codificada de otra máquina de Turing (⟨M⟩)* y una *cadena de entrada (w)* para esa máquina M. La descripción de M incluye sus estados, alfabeto, y función de transición.
- **Simulación:** La MT U *ejecuta la máquina M* sobre la entrada w, basándose en la descripción que recibió.
- **Comportamiento:** La MT U *se comporta de la misma manera* que la máquina M que está simulando. Si M acepta w, entonces U también acepta (simulando la aceptación de M). Si M rechaza w o entra en un bucle infinito, U hará lo mismo (simulando el comportamiento de M).
- **Programa Almacenado:** La MT universal incorpora el concepto de un *programa almacenado* (la descripción de la máquina M) que puede ser interpretado y ejecutado.
- **Importancia:** La existencia de la MT universal es fundamental en la teoría de la computabilidad. Demuestra que *una única máquina puede realizar cualquier cálculo* que cualquier otra máquina de Turing pueda realizar, lo que refuerza la Tesis de Church-Turing. También es crucial para probar la *indecibilidad del Problema de la Parada (Halting Problem).*

En esencia, una MT universal es una *modelización teórica de una computadora programable de propósito general.*



Ejercicio 2

Explicar cómo enumeraría los números naturales pares, los números enteros, los números racionales (o fraccionarios), y las cadenas de Σ* siendo Σ = {0, 1}.

> Definiciones que pueden servir para entender
>
> **Biyección:** Correspondencia exacta entre los elementos de dos conjuntos. Es decir, cada elemento de un conjunto tiene un único y exacto compañero en el otro conjunto, y viceversa.
> **Función Biyectiva:** Una función $f: A \to B$ es biyectiva si se cumple dos condiciones:
>
> 1. Es **Inyectiva**, es decir, cada elemento del conjunto de salida B tiene como mucho un elemento en el conjunto de entrada A.
> 2. Es **Sobreyectiva**, es decir, cada elemento de B tiene al menos un elemento de A que lo genera. No quedan elementos de B sin usar.

Números naturales pares:

El conjunto de los números naturales pares $\mathbb{N}_p = \{0, 2, 4, 6, 8, \dots\}$, se puede enumerar estableciendo una biyección con el conjunto de los números naturales $\mathbb{N}$. La función que podemos utilizar es $f(n) = 2n,\ para\ n \in \mathbb{N}$. Esto se puede visualizar como la siguiente lista ordenada:

- 0 (corresponde a n=0), 2 (corresponde a n=1), 4 (corresponde a n=2), 6 (corresponde a n=3), y así sucesivamente.

Números enteros:

Para construir una enumeración de este conjunto podemos definir una función $f: \mathbb{N} \to \mathbb{Z}$, donde $\mathbb{N} = 0, 1, 2, \dots$ es el conjunto de los números naturales (incluyendo el cero) y $\mathbb{Z}$ es el conjunto de los números enteros, de la siguiente manera:

- Si $n = 0$, entonces $f(n) = 0$.
- Si n es un número natural impar, entonces $f(n) = \frac{n+1}{2}$.
- Si n es un número natural par y $n > 0$, entonces $f(n) = -\frac{n}{2}$.
  Esta función genera la secuencia de los números enteros:
- $f(0) = 0$
- $f(1) = \frac{1+1}{2} = 1$
- $f(2) = -\frac{2}{2} = -1$
- $f(3) = \frac{3+1}{2} = 2$

- $f(4) = -\frac{4}{2} = -2$
- $f(5) = \frac{5+1}{2} = 3$
- $f(6) = -\frac{6}{2} = -3$
- Y así sucesivamente.

Números racionales (o fraccionarios):

La clave para enumerar los racionales es organizar todos los posibles pares de enteros (que representan el numerador y el denominador) de una manera sistemática y luego asignarles un número natural único, evitando repeticiones.

1. Considerar pares de enteros (a, b) donde b ≠ 0: Cada número racional puede ser expresado como una fracción $\frac{a}{b}$, donde a es un entero y b es un entero diferente de cero.
2. *Organizar por la suma del valor absoluto:* Podemos empezar organizando estos pares según la suma del valor absoluto del numerador y el denominador ($|a| + |b|$).
   - Suma = 1: (0, 1), (1, 0) - ignoramos (1, 0) porque el denominador no puede ser 0. Fracciones: $\frac{0}{1} = 0$.
   - Suma = 2: (0, 2), (2, 0), (1, 1), (-1, 1), (1, -1), (-1, -1). Ignoramos (2, 0) y (-2, 0). Fracciones (simplificando y sin repeticiones aún): $\frac{0}{2} = 0$ (ya contado), $\frac{1}{1} = 1$, $\frac{-1}{1} = -1$, $\frac{1}{-1} = -1$ (ya contado), $\frac{-1}{-1} = 1$ (ya contado). Nuevos: $1, -1$.
   - Suma = 3: (0, 3), (3, 0), (1, 2), (-1, 2), (2, 1), (-2, 1), (1, -2), (-1, -2), (2, -1), (-2, -1), (3, -1), (-3, 1), etc. y así sucesivamente.
3. *Ordenar dentro de cada suma:* Dentro de cada grupo con la misma suma $|a| + |b|$, podemos ordenarlos por el valor del numerador (a), y luego considerar el signo.
4. *Eliminar repeticiones:* A medida que generamos las fracciones $\frac{a}{b}$, debemos asegurarnos de no incluir repeticiones. Una forma de hacer esto es considerar solo las fracciones en su forma irreducible (donde el máximo común divisor de $|a|$ y $|b|$ es 1, y manejar el signo apropiadamente).
5. *Intercalar positivos y negativos:* Después de generar las fracciones positivas en forma irreducible, podemos intercalar sus contrapartes negativas y también incluir el cero.

Las cadenas de Σ* siendo Σ = {0, 1} se pueden enumerar utilizando el orden canónico. Este orden primero lista las cadenas por su longitud, en orden ascendente, y luego, para las cadenas de la misma longitud, se utiliza el orden lexicográfico. Para Σ = {0, 1}, la enumeración comenzaría así:

- λ (cadena vacía, longitud 0) 0, 1 (longitud 1) 00, 01, 10, 11 (longitud 2) 000, 001, 010, 011, 100, 101, 110, 111 (longitud 3) y así sucesivamente.

Ejercicio 3

Dar la idea general de cómo sería una MT que, teniendo como cadena de entrada un número natural i, genera la i-ésima fórmula booleana satisfactible según el orden canónico. *Comentario: asumir que existen una MT M_1 que determina si una cadena es una fórmula booleana, y una MT M_2 que determina si una fórmula booleana es satisfactible.*

La Máquina de Turing M que buscamos operará generando secuencialmente todas las posibles cadenas de símbolos en el orden canónico. Para cada cadena generada, M verificará si cumple dos condiciones: primero, si es una fórmula booleana válida (utilizando la hipotética M_1), y segundo, si dicha fórmula es satisfactible (utilizando la hipotética M_2). Se mantendrá un contador para rastrear cuántas fórmulas booleanas satisfactibles se han encontrado hasta el momento. Cuando el contador alcance el valor del número natural de entrada i, la MT M se detendrá y producirá la fórmula correspondiente.

Pasos Detallados de la MT M:

1. *Entrada:* La MT M recibe como entrada un número natural i.
2. *Inicialización del Contador:* M inicializa un contador interno (que puede ser implementado como una sección en su cinta de trabajo) a cero. Llamemos a este contador j.
3. *Generación de Cadenas en Orden Canónico:* M comienza a generar todas las posibles cadenas de símbolos sobre un alfabeto apropiado (que incluya los símbolos necesarios para construir fórmulas booleanas, como variables, conectores lógicos ($\wedge, \vee, \neg$), paréntesis, etc.) siguiendo el orden canónico.
4. *Verificación de Fórmula Booleana (M_1):* Para cada cadena w generada, M debe simular el comportamiento de la MT M_1 sobre w.
   - Si M_1 acepta w (determina que w es una fórmula booleana válida), entonces M continúa con el siguiente paso.
   - Si M_1 rechaza w (determina que w no es una fórmula booleana válida), entonces M descarta w y vuelve al paso 3 para generar la siguiente cadena en orden canónico.
5. *Verificación de Satisfactibilidad (M_2):* Si la cadena w ha sido identificada como una fórmula booleana por la simulación de M_1, entonces M debe simular el comportamiento de la MT M_2 sobre w.
   - Si M_2 acepta w (determina que la fórmula booleana w es satisfactible), entonces M incrementa su contador j en uno.
   - Si M_2 rechaza w (determina que la fórmula booleana w no es satisfactible), entonces M descarta w y vuelve al paso 3 para generar la siguiente cadena en orden canónico.
6. *Comparación con el Número de Entrada:* Después de cada incremento del contador j (cuando se encuentra una fórmula booleana satisfactible), M compara el valor de j con el número natural de entrada i.
7. *Aceptación y Salida:*

- Si el valor de j es igual a i, significa que M ha encontrado la i-ésima fórmula booleana satisfactible en el orden canónico. En este punto, M detiene su computación y deja la fórmula w en su cinta de trabajo como resultado. Se podría definir un estado de aceptación q_A para indicar la terminación exitosa.
- Si el valor de j aún no es igual a i, M vuelve al paso 3 para generar la siguiente cadena en orden canónico y repite el proceso.

Ejercicio 4

Sea M_1 una MT que genera en su cinta de salida todas las cadenas de un lenguaje L. Dar la idea general de cómo sería una MT M_2 que, usando M_1, acepte una cadena w sii w ∈ L.

La idea general de cómo sería una MT M_2 que, usando M_1, acepte una cadena w sii $w \in L$ se basa en simular la MT generadora M_1 y comparar su salida con la entrada w de M_2.

M_2 podría funcionar de la siguiente forma:

1. *Entrada:* La MT M_2 recibe una cadena w en su cinta de entrada.
2. *Simulación de M_1:* M_2 comienza a simular la ejecución de M_1. Como M_1 es una MT generadora, escribirá secuencialmente las cadenas del lenguaje L en su cinta de salida.
3. *Comparación:* A medida que M_1 genera y escribe una cadena v en su cinta de salida, M_2 compara esta cadena v con su propia cadena de entrada w.
4. *Aceptación:*
   - Si en algún momento la cadena v generada por M_1 es idéntica a la cadena w de entrada de M_2, entonces M_2 entra en su estado de aceptación (q_A) y se detiene. Esto indica que w pertenece al lenguaje L generado por M_1.
5. *Rechazo (Implícito):*
   - Si la MT generadora M_1 termina su ejecución sin haber generado la cadena w, entonces M_2 debe eventualmente rechazar w. La forma en que M_2 detecta esta situación depende de cómo se define la terminación de M_1. Si M_1 tiene un estado de parada, M_2 puede alcanzar un estado de rechazo (q_R) si M_1 termina sin haber generado w.

Es importante considerar que si el lenguaje L es infinito, la MT generadora M_1 podría no detenerse nunca. En este caso, si la cadena w no pertenece a L, M_2 continuará simulando M_1 indefinidamente sin encontrar una coincidencia y, por lo tanto, nunca aceptará. En el contexto de la definición de lenguajes recursivamente enumerables (RE), esta es una posibilidad para cadenas que no pertenecen al lenguaje.

Ejercicio 5

El lenguaje L_U = {(<M>, w) | M acepta w} se conoce como lenguaje universal, y representa el problema general de aceptación. Probar que L_U ∈ RE. *Ayuda: construir una MT que acepte L_U.*

La MT U tomará como entrada la codificación de otra MT M, denotada como $< M >$, y una cadena de entrada w para M. U simulará la ejecución de M sobre w. Si esta simulación resulta en que M acepta w, entonces U también aceptará la entrada $(< M >, w)$. Si M rechaza w o no se detiene, U no necesariamente tiene que rechazar; para que L_U sea RE, basta con que U acepte cuando M acepta.

Pasos de la MT U:

1. *Entrada:* La MT U recibe como entrada una cadena que representa el par $(< M >, w)$.
2. *Interpretación de la Codificación:* U interpreta la primera parte de su entrada para obtener la descripción de la MT M (su conjunto de estados, su alfabeto de cinta, su función de transición, su estado inicial y sus estados de aceptación y rechazo). U también identifica la segunda parte de su entrada como la cadena w que M procesará.
3. *Simulación de M sobre w:* U utiliza su propia cinta para simular la cinta de M, el estado actual de M y la posición de la cabeza de lectura/escritura de M.
   - Inicialmente, la cinta simulada de M contendrá la cadena w, la cabeza estará en la posición inicial, y el estado actual de M será su estado inicial.
   - U consultará la función de transición de M (que obtuvo de $< M >$) para determinar el siguiente estado, el símbolo a escribir, y la dirección en la que se moverá la cabeza, basándose en el estado actual simulado de M y el símbolo bajo la cabeza simulada.
   - U actualizará el estado simulado, el contenido de la cinta simulada y la posición de la cabeza simulada de acuerdo con la transición.
   - U repetirá este proceso, simulando cada paso de la computación de M sobre w.
4. *Condición de Aceptación:*
   - Si en algún momento durante la simulación, la MT M entra en su estado de aceptación (q_A), entonces la MT U también entra en su estado de aceptación (q_A) y se detiene. Esto indica que M ha aceptado w, y por lo tanto, $(< M >, w)$ pertenece a L_U.
5. *No Aceptación:*
   - Si M entra en su estado de rechazo (q_R), entonces U no necesita aceptar. En este caso, U podría detener la simulación y no aceptar, o incluso entrar en un estado de rechazo propio (aunque esto no es estrictamente necesario para probar que $L_U \in RE$).
   - Si M nunca entra en un estado de aceptación o rechazo (es decir, M entra en un bucle infinito), entonces la simulación por U continuará indefinidamente, y U nunca entrará en su estado de aceptación.

De acuerdo con la definición de lenguaje recursivamente enumerable, un lenguaje L es RE si existe una MT que lo reconoce, es decir, que acepta todas las cadenas $w \in L$, y para $w \notin L$, la MT puede rechazar o no detenerse. En nuestro caso, la MT U que hemos descrito acepta precisamente cuando M acepta w, que es la condición para que $(< M >, w) \in L_U$. Si M no acepta w, U no necesita aceptar. Por lo tanto, U es una MT que acepta L_U, lo que demuestra que $\mathbf{L_U \in RE}$

Ejercicio 6

Una función f : A ⟶ B es total computable sii existe una MT M_f que la computa para todo elemento a ∈ A. Sea la función f_{01} : Σ* ⟶ {0, 1} tal que:

- f_{01}(v) = 1, si v = (<M>, w) y M para a partir de w.
- f_{01}(v) = 0, si v = (<M>, w) y M no para a partir de w o bien v ≠ (<M>, w).
  Probar que f_{01} no es total computable. *Ayuda: ¿con qué problema se relaciona dicha función?*

La función f_{01} se relaciona directamente con el problema de la parada (halting problem) que busca determinar dada una Máquina de Turing M y una entrada w, si M se detiene a partir de w. El lenguaje que representa este problema se conoce como HP, definido como $HP = (< M >, w) \mid M \text{ se detiene a partir de } w$.

Vamos a demostrar por contradicción que f_{01} no es total computable.

Supongamos que f_{01} es total computable. Esto significa que existe una Máquina de Turing $M_{f_{01}}$ que, dada cualquier cadena v, siempre se detiene y produce:

- 1, si v es de la forma $(< M >, w)$ y M se detiene a partir de w.
- 0, si v es de la forma $(< M >, w)$ y M no se detiene a partir de w, o si v no tiene la forma de una codificación válida de un par $(< M >, w)$.

Si tuviéramos una MT $M_{f_{01}}$ que calcula f_{01}, podríamos usarla para construir una MT que decide el problema de la parada (HP). Una MT que decide un lenguaje siempre se detiene y acepta si la entrada pertenece al lenguaje, y se detiene y rechaza si no pertenece.

Consideremos la construcción de una MT M_{HP} que toma como entrada $(< M >, w)$ y funciona de la siguiente manera:

1. M_{HP} recibe $(< M >, w)$ en su cinta de entrada.
2. M_{HP} simula la computación de $M_{f_{01}}$ con la entrada $(< M >, w)$. Como asumimos que f_{01} es total computable, $M_{f_{01}}$ siempre se detendrá y producirá un resultado (0 o 1).

3. Si la salida de $M_{f_{01}}$ es 1, entonces M_{HP} acepta $(< M >, w)$ (porque $f_{01}((< M >, w)) = 1$ implica que M se detiene a partir de w).
4. Si la salida de $M_{f_{01}}$ es 0, entonces M_{HP} rechaza $(< M >, w)$ (porque $f_{01}((< M >, w)) = 0$ implica que M no se detiene a partir de w, o que la entrada no era válida, en cuyo caso no pertenece a HP tal como se define para pares MT-cadena válidos).

La MT M_{HP} que hemos construido siempre se detiene (porque $M_{f_{01}}$ siempre se detiene) y decide correctamente si $(< M >, w)$ pertenece al lenguaje HP. Es decir, M_{HP} decide el problema de la detención.

Sin embargo, *el Teorema de Indecibilidad del Halting Problem establece que el lenguaje HP no es recursivo* (no existe una MT que lo decida). Esta contradicción surge de nuestra suposición inicial de que f_{01} es total computable.

Por lo tanto, concluimos que la función f_{01} no es total computable. Si existiera una MT que calculara f_{01} para todas las entradas, podríamos decidir el problema de la detención, lo cual sabemos que es imposible. La función f_{01} representa precisamente la capacidad de decidir si una MT se detiene o no, y como el problema de la detención es indecidible, dicha función no puede ser total computable.

Ejercicio 7

Responder breve y claramente cada uno de los siguientes incisos (en todos los casos, las MT mencionadas tienen una sola cinta):

1. Probar que se puede decidir si una MT M, a partir de la cadena vacía λ, escribe alguna vez un símbolo no blanco. *Ayuda: ¿Cuántos pasos puede hacer M antes de entrar en un loop?*

Sí se puede decidir si una MT M, a partir de la cadena vacía λ, escribe alguna vez un símbolo no blanco. Si M no escribe un símbolo no blanco dentro de un cierto número de pasos, comenzará a repetir configuraciones (estado, posición del cabezal, contenido de la cinta). Dado que el número de estados es finito, y si consideramos un fragmento acotado de la cinta alrededor de la posición inicial (ya que inicialmente la cinta está en blanco y no nos interesa más que una única posición para ir escribiendo ahí), el número de configuraciones posibles antes de escribir un símbolo no blanco también es finito.

Podemos simular M a partir de λ durante un número suficientemente grande de pasos. Si en ese tiempo escribe un símbolo no blanco, decidimos que sí. Si no lo hace y no ha parado, entonces ha entrado en un bucle sin escribir un símbolo no blanco, y decidimos que no.

2. Probar que se puede decidir si una MT M que sólo se mueve a la derecha, a partir de una cadena w, para, *Ayuda: ¿Cuántos pasos puede hacer M antes de entrar en un loop?*

Sí se puede decidir si una MT M que sólo se mueve a la derecha, a partir de una cadena w, para. Como M sólo se mueve a la derecha, eventualmente alcanzará la parte en blanco de la cinta. Una vez en la parte en blanco, su comportamiento dependerá únicamente de su estado actual. Si en algún momento M entra en un estado que ya ha visitado mientras está leyendo el símbolo blanco, entrará en un bucle infinito.

Podemos simular M a partir de w. Si M para, decidimos que sí. Si después de un número suficientemente grande de pasos (relacionado con el número de estados de M) no ha parado, y se encuentra en la región de blancos y ha repetido un estado, podemos concluir que entrará en un bucle infinito.

3. Probar que se puede decidir si dada una MT M, existe una cadena w a partir de la cual M para en a lo sumo 10 pasos. *Ayuda: ¿Hasta qué tamaño de cadenas hay que chequear?*

Sí se puede decidir si dada una MT M, existe una cadena w a partir de la cual M para en a lo sumo 10 pasos. En 10 pasos, el cabezal de M puede alcanzar celdas que estén a una distancia de a lo sumo 10 de la posición inicial. Por lo tanto, sólo necesitamos considerar las cadenas de entrada de longitud a lo sumo 10. Para cada una de estas finitas cadenas posibles sobre el alfabeto de M, podemos simular la ejecución de M durante a lo sumo 10 pasos. Si para alguna de estas cadenas M se detiene dentro de ese límite de pasos, entonces la respuesta es sí. Si para ninguna de las cadenas de longitud hasta 10 M se detiene en 10 pasos o menos, entonces la respuesta es no.

4. ¿Se puede decidir si dada una MT M, existe una cadena w de a lo sumo 10 símbolos a partir de la cual M para? Justificar la respuesta.

Sí, se puede decidir si dada una MT M, existe una cadena w de a lo sumo 10 símbolos (de su alfabeto Γ) a partir de la cual M para.

La justificación se basa en el hecho de que solo hay un número finito de posibles cadenas de entrada de una longitud máxima de 10 símbolos sobre el alfabeto finito Γ de la Máquina de Turing M. Si el alfabeto Γ tiene k símbolos, entonces el número de cadenas de longitud l es k^l. Por lo tanto, el número total de cadenas de longitud a lo sumo 10 es $\sum_{i=0}^{10} k^i$, que es un número finito.

Para cada una de estas finitas cadenas de entrada, podemos simular la ejecución de la Máquina de Turing M. Si M, al ejecutarse con una entrada w de longitud a lo sumo 10, no se detiene, entonces en algún momento debe entrar en un bucle, lo que significa que repetirá una configuración en su ejecución.

¿Qué tan grande debe ser este número de pasos para detectar un bucle? Consideremos el número de estados de M (denotémoslo por $|Q|$) y el alfabeto de M (denotémoslo por $|\Gamma|$). Para una entrada de longitud a lo sumo 10, si la máquina no se ha detenido después de haber realizado un número de pasos que excede un cierto límite basado en $|Q|$ y una porción relevante de la cinta (que podemos acotar en función del número de pasos simulados), entonces se puede argumentar que la máquina ha entrado en un bucle.

Un límite superior para el número de configuraciones únicas que una MT puede pasar antes de entrar en un bucle, si se limita a un espacio de K celdas, está dado por $|Q| \times K \times |\Gamma|^K$. Aunque no tenemos un límite espacial fijo de antemano, si la máquina no se detiene en un número de pasos razonablemente grande (por ejemplo, un número que considere todas las combinaciones de estados y símbolos en una región de la cinta que ha sido alcanzada), podemos inferir un bucle.