

FTC - PRÁCTICA

PRACTICA 1 - LA MÁQUINA DE TURING (MT)

Ejercicio 1. Responder breve y claramente los siguientes incisos:

1. *¿En qué se diferencia un problema de búsqueda de un problema de decisión?*

Se diferencian en que un problema de búsqueda espera obtener como respuesta una solución a dicho problema. Mientras que un problema de decisión requerirá obtener como respuesta si el problema puede ser resuelto o no.

2. *¿Por qué en el caso de los problemas de decisión, podemos referirnos indistintamente a problemas y lenguajes?*

Porque depende del lenguaje aceptado si el problema puede ser resuelto o no.

Existe una correspondencia directa entre ambos conceptos, ya que un lenguaje es un conjunto de cadenas (sobre un alfabeto) que representan las entradas donde la respuesta es "sí".

3. *El problema de satisfactibilidad de las fórmulas booleanas, en su forma de decisión, es: "Dada una fórmula φ , ¿existe una asignación A de valores de verdad que la hace verdadera?" Enunciar el problema de búsqueda asociado.*

¿Cuál es la asignación A de valores que la hace verdadera?

4. *Otra visión de MT es la que genera un lenguaje (visión generadora). En el caso del problema del inciso anterior, ¿qué lenguaje generaría la MT de visión generadora que resuelve el problema?*

Generaría las cadenas de asignaciones A que hagan verdadera la fórmula.

5. *¿Qué postula la Tesis de Church-Turing?*

Postula que todo dispositivo computacional físicamente realizable puede ser simulado por una MT.

6. *¿Cuándo dos MT son equivalentes? Y cuándo dos modelos de MT son equivalentes?*
Dos MT son equivalentes cuando aceptan el mismo lenguaje, es decir que resuelven el mismo problema.

Dos modelos de MT son equivalentes cuando dada una MT de un modelo existe una MT equivalente de otro. Ej: MT con una cinta y MT con varias cintas.

Ejercicio 2. Dado el alfabeto $\Sigma = \{0, 1\}$:

1. *Obtener el conjunto Σ^* y el lenguaje incluido en Σ^* con cadenas de al menos 2 símbolos.*

$\Sigma^* = \{0, 1, 00, 11, 01, 10, 000, 111, \dots\}$

$L = \{0, 1, 00, 11, 01, 10\}$

2. *Sea el lenguaje $L = \{0^n 1^n \mid n \geq 0\}$. Obtener los lenguajes $\Sigma^* \cap L$, $\Sigma^* \cup L$ y L^C respecto de Σ^* .*

$\Sigma^* \cap L = L$

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

$L(c)$ respecto de $\Sigma^* = \{000, 001, 010, 100, 011, 110, 101, 111, \dots\}$

Ejercicio 3.

En clase se mostró una MT no determinística (MTN) que acepta las cadenas de la forma $ha(n)$ o $hb(n)$, con $n \geq 0$. Construir (describir la función de transición) una MT determinística (MTD) equivalente.

$$\delta(q_0, h) = (q_1, h, R)$$

$$\delta(q_1, a) = (q_2, a, R)$$

$$\delta(q_1, b) = (q_3, b, R)$$

$$\delta(q_2, a) = (q_2, a, R)$$

$$\delta(q_2, b) = (q_R, b, S)$$

$$\delta(q_2, B) = (q_A, B, S)$$

$$\delta(q_3, b) = (q_3, b, R)$$

$$\delta(q_3, a) = (q_R, a, S)$$

$$\delta(q_3, B) = (q_A, B, S)$$

Ejercicio 4.

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

Voy a utilizar tres cintas para procesar la entrada de manera más efectiva:

- Cinta 1: Contiene la entrada $a^n b^n c^n$.
- Cinta 2: Contará los caracteres 'a' y 'b'.
- Cinta 3: Contará los caracteres 'b' y 'c'.

Funcionamiento:

1. Verificar la estructura correcta:
 - . Asegurarse de que la cadena sigue el patrón $a^n b^n c^n$ (sin desorden).
 - . Si la cadena está vacía ($n=0$), aceptar inmediatamente.
2. Mover las 'a' a la segunda cinta:
 - . Mientras leo 'a' en la cinta 1, copio un marcador ('X') en la cinta 2 para llevar la cuenta de n .
3. Mover las 'b' a la tercera cinta y comparar:
 - . Luego, leo las 'b' en la cinta 1 y las copio como 'Y' en la cinta 3.
 - . Simultáneamente, verifico que hay exactamente n 'X' en la cinta 2 (una coincidencia por cada 'b').
4. Verificar la cantidad de 'c':
 - . Finalmente, recorro los 'c' en la cinta 1 y verifico que haya exactamente n 'Y' en la cinta 3.
5. Aceptar o rechazar:
 - . Si al final todas las cantidades coinciden (y la cinta original quedó vacía), aceptar.
 - . Si hay desajustes en las cantidades o en el orden, rechazar.

Estados:

- q_0 (Inicio):

- . Mueve el cabezal de la cinta 1 hasta el primer símbolo 'a'.
- . Si la cinta está vacía, aceptar ($n=0$).
- . Transición a q_1 si encuentra 'a'.
 - q_1 (Marcar 'a' y copiar a la cinta 2):
- . Mientras haya 'a' en la cinta 1, reemplaza cada 'a' con 'X' y copia un marcador ('X') en la cinta 2.
- . Cuando encuentra el primer 'b', pasa a q_2 .
 - q_2 (Contar 'b' y copiar a la cinta 3):
- . Mientras haya 'b' en la cinta 1, reemplaza cada 'b' con 'Y' y copia un marcador ('Y') en la cinta 3.
- . Simultáneamente, verifica que haya exactamente un 'X' en la cinta 2 por cada 'b'.
- . Cuando encuentra el primer 'c', pasa a q_3 .
 - q_3 (Verificar 'c' y comparar con la cinta 3):
- . Mientras haya 'c' en la cinta 1, verifica que haya exactamente un 'Y' en la cinta 3 por cada 'c'.
- . Si al final de los 'c', las cintas 2 y 3 están vacías y la cinta 1 llega al blanco, pasa a q_A (aceptación).
- . Si hay desajustes, ir a q_R (rechazo).
 - q_A (Aceptar):
 - . Si la verificación fue exitosa y todas las cintas terminaron de procesarse correctamente.
 - q_R (Rechazar):
 - . Si hay desajustes en las cantidades de 'a', 'b' o 'c', o si la estructura es incorrecta.

Ejercicio 5.

Explicar cómo una MT sin el movimiento S (el no movimiento) puede simular (ejecutar) otra que sí lo tiene.

Puede simularlo utilizando un estado adicional que lo único que haga es mantener el símbolo que esté escrito y moviendo a la izquierda. Entonces, al querer mantener la posición, pasará al nuevo estado (ej. q_{VOLVER}), manteniendo el símbolo que esté escrito y moviendo a la derecha.

Ej:

$$\delta(q_1, a) = (q_{VOLVER}, a, R)$$

$$\delta(q_{VOLVER}, a) = (q_1, a, L)$$

$$\delta(q_{VOLVER}, b) = (q_1, b, L)$$

.....

Ejercicio 6.

En clase se construyó una MT con 2 cintas que acepta $L = \{w \mid w \in \{a, b\}^ \text{ y } w \text{ es un palíndromo}\}$. Construir una MT equivalente con 1 cinta. Ayuda: la solución que vimos para aceptar el lenguaje de las cadenas $a^n b^n$, con $n \geq 1$, puede ser un buen punto de partida.*
 Voy a utilizar 4 pistas en la misma cinta.

El separador será 'X'.

PISTA 1	A	B	B	A
---------	---	---	---	---

PISTA 2	X			
PISTA 3	A B B A			
PISTA 4	Z			

q0: Recorrer hasta el final y colocar una X.
 Marcar: Recorre hasta el final y coloca una Z.
 qVolverInicio: Recorrer hasta el inicio.
 qLeer: lee un carácter y cambia las A por C y las B por D.
 qCopiarA: avanza hasta un espacio vacío y escribe una A.
 qCopiarB: avanza hasta un espacio vacío y escribe una B.
 qVolver: desplaza a la izquierda hasta encontrar una C o D.
 qChequear1: lee un carácter y coloca una Y.
 qAvanzarCheckA: avanza hasta una X o Z y manda a chequear por una A.
 qAvanzarCheckB: avanza hasta una X o Z y manda a chequear por una B.
 qChequear2_A: avanza hasta una X o Z y chequea que haya una A. La cambia por una Z.
 qChequear2_B: avanza hasta una X o Z y chequea que haya una B. La cambia por una Z.
 qVolverCheck: desplaza a la izquierda hasta encontrar una Y.

	a	b	C	D	X	Y	Z	B
q0	q0, a, R	q0, b, R						qVolverInicio, X, L
qVolverInicio	qVolverInicio, a, L	qVolverInicio, b, L	qVolverInicio, C, L	qVolverInicio, D, L	qVolverInicio, X, L			qLeer, B, R
qMarcar	qMarcar, a, R	qMarcar, b, R						qVolverInicio, Z, L
qLeer	qCopiarA, C, R	qCopiarB, D, R			qMarcar, X, R			
qCopiarA	qCopiarA, a, R	qCopiarA, b, R			qCopiarA, X, R			qVolver, A, L
qCopiarB	qCopiarB, a, R	qCopiarB, b, R			qCopiarB, X, R			qVolver, B, L
qVolver	qVolver, a, L	qVolver, b, L	qLeer, C, R	qLeer, D, R	qVolver, X, L			
qChequear1			qAvanzarCheckA, Y, R	qAvanzarCheckB, Y, R	qA, X, S			
qAvanzarCheckA			qAvanzarCheckA, C, R	qAvanzarCheckA, D, R	qAvanzarCheckA, X, R		qChequear2_A, Z, L	
qAvanzarCheckB			qAvanzarCheckB, C, R	qAvanzarCheckB, D, R	qAvanzarCheckB, X, R		qChequear2_B, Z, L	
qChequear2_A	qVolverCheck, Z, L	qR, b, S						
qChequear2_B	qR, a, S	qVolverCheck, Z, L						

Ejercicio 7.

Construir una MT que calcule la resta de dos números. Ayuda: se puede considerar la idea de solución propuesta en clase.

Aceptare 2 números codificados en unario separados por un 0.

q0: lee un 1 y lo cambia por una X.

qIrNum2: avanza hasta el 0.

qRestar: avanza hasta el primer 1 y lo cambia por una X. Si encuentra un B pasa al estado qFin.

qIrNum1: vuelve hasta el 0.

qVolverInicio: vuelve hasta el primer 1.

qFin: vuelve hasta el 0, lo cambia por una X y acepta.

	0	1	X	B
q0	qR, 0, S	qIrNum2, X, R		
qIrNum2	qRestar, 0, R	qIrNum2, 1, R		
qRestar		qIrNum1, X, L	qRestar, X, R	qFin, B, L
qIrNum1	qVolverInicio, 0, L		qIrNum1, X, L	
qVolverInicio		qVolverInicio, 1, L	q0, X, R	
qFin	qA, X, S		qFin, X, L	

Ejercicio 8.

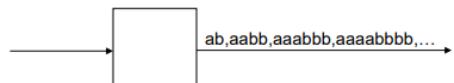
Construir una MT que genere todas las cadenas de la forma $a^n b^n$, con $n \geq 1$. Ayuda: se puede considerar la idea de solución propuesta en clase.

Visión de MT generadora (genera en una cinta de salida todas las cadenas del lenguaje que acepta)

Problema: construir una MT que genere todas las cadenas de la forma $a^n b^n$, con $n \geq 1$. Es decir, en una cinta especial de salida debe **generar las cadenas ab, aabb, aaabbb, aaaabbbb, etc.**

Idea general:

- (1) $i := 1$
- (2) imprimir i veces a, imprimir i veces b, e imprimir una coma
- (3) $i := i + 1$ y volver a (2)



Idea: 2 cintas.

Cinta 1: escribir cadenas $a^n b^n c^n$,

Cinta 2: contador.

Estado para escribir As, por cada '1' que haya en la segunda cinta escribe una A.

Pasa a estado de escribir B.

Estado para escribir Bs, por cada '1' que haya en la segunda cinta escribe una B.

Pasa a estado de incrementar contador.

Estado de incrementar contador agrega un 1 en la segunda cinta y escribe una ',' al final de la cadena. Pasa a estado de escribir A.

PRÁCTICA 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 enumerables no recursivos y los lenguajes no recursivamente numerables?*

Los lenguajes recursivos son aquellos que tienen una MT que lo acepta y siempre para (lo decide). Por su parte los recursivamente enumerables tienen una MT que los resuelve donde si hay al menos una cadena correcta deberá parar por “si”, mientras que si no la hay podría parar por “no” o loopear, es decir que no siempre para.

Los no recursivamente enumerables son aquellos para los cuales no existe una Máquina de Turing que pueda reconocer las palabras del lenguaje.

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

$R \subseteq RE$ por definición.

$L_1 \in RE$, por lo tanto $L_1(\text{complemento}) \in CO-RE$. $CO-RE \cup RE \subseteq \mathcal{L}$, por lo tanto $RE \subseteq \mathcal{L}$.

Entonces $R \subseteq RE \subseteq \mathcal{L}$.

3. *Dijimos en clase que el hecho de que si L es recursivo entonces $L(\text{complemento})$ también lo es, significa en términos de problemas que si un problema es decidable 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 intersección de 2 lenguajes recursivos seguirá siendo recursiva. Por lo que si existe una MT M_1 que decide L_1 , y existe una MT M_2 que decide L_2 , también existe una MT M que decide $L_1 \cap L_2$.

4. *Explicar por qué no es correcta la siguiente prueba de que si $L \in RE$, también $L(\text{complemento}) \in RE$: dada una MT M que acepta L , entonces la MT M' , igual que M pero con los estados finales permutados, acepta L^c .*

Porque una máquina que acepte un lenguaje RE podría loopear al tener $w \notin L$. Por lo que no alcanzaría con permutar las respuestas porque puede no haber.

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

Todos los lenguajes de la clase CO-RE tienen MT que los aceptan ya que todos son complemento de los que pertenecen a la clase RE.

No, ya que al ser complementos de RE pueden loopear. Sólo serán decidibles aquellos que sean RE y CO-RE al mismo tiempo ya que $RE \cap CO-RE = R$, y los $L \in R$ son decidibles.

6. *Probar que el lenguaje Σ^* de todas las cadenas y el lenguaje vacío \emptyset son recursivos.*

Alcanza con plantear la idea general. Ayuda: encontrar MT que los decidan.

Lenguaje \emptyset :

Construir una MT que ante cualquier entrada responda qR.

Lenguaje Σ^* :

Construir una MT que ante cualquier entrada w responda qA.

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).

Construir una MT que ante cualquier entrada w lea cada carácter, sin importar cual, uno por uno y se desplace a la derecha hasta encontrar un B y acepte (qA). Al ser un lenguaje finito en algún momento llegará a su fin sin quedarse loopando.

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 copiaría de a un carácter a medida que voy ejecutando M_1 , para de esta manera no perder tiempo copiando toda la cadena si quizás M_1 rechaza al leer el primer carácter.

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

Volvería a buscar los caracteres a la cinta 1 y los sobreescibiría en la cinta 2. También respetando la implementación de la copia propuesta en 1. copiando de a un carácter, por lo tanto, sobreescribiendo de a un carácter.

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.

Si existe una MT M_1 que decide L_1 , y existe una MT M_2 que decide L_2 , también existe una MT M que decide $L_1 \cup L_2$.

Si $L_1 \in R$ y $L_2 \in R$, entonces $L_1 \cup L_2 \in R$.

Idea: Construir una MT M que ejecute secuencialmente M_1 y luego M_2 . Alcanza con que una de las 2 responda "sí" para que M responda "sí". Y M solo responderá "no" en el caso en que ambas M_1 y M_2 respondan "no".

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 .

Si existe una MT M_1 que acepta L_1 , y existe una MT M_2 que acepta L_2 , también existe una MT M que acepta $L_1 \cap L_2$.

Si $L_1 \in RE$ y $L_2 \in RE$, entonces $L_1 \cap L_2 \in RE$.

Idea: Construir una MT M que ejecute M_1 y M_2 "en paralelo" (alternando un paso de M_1 con uno de M_2). M aceptará la entrada w si ambas M_1 y M_2 aceptan w . Si alguna de ellas loopea, M loopeará. Y si ambas rechazan, M rechazará.

Ejercicio 4.

Sean L_1 y L_2 dos lenguajes recursivamente enumerables de números naturales codificados en unaryo (por ejemplo, el número 5 se representa con 11111). Probar que también es recursivamente numerable el lenguaje $L = \{x \mid x \text{ es un número natural codificado en unaryo, y existen } y, z, \text{ tales que } y + z = x, \text{ con } y \in L_1, z \in L_2\}$.

Ayuda: la prueba es similar a la vista en clase, de la clausura de la clase RE con respecto a la operación de concatenación.

Construir una MT M que decida el lenguaje L_1+L_2 .

Dado un input w (un número codificado en unaryo), la idea es dividir en 2 ese número y chequear que la primer parte corresponda a un número en L_1 y que la segunda parte corresponda a un número en L_2 .

Dado el input w , M hace:

1. M ejecuta M_1 a partir de los primeros 0 símbolos de w , y M_2 a partir de los últimos n símbolos de w . Si en ambos casos se acepta, entonces M acepta.
2. 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.
3. 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.

M ejecuta M_1 y M_2 en paralelo. De esta manera puede probar todas las combinaciones hasta encontrar una que sea correcta y acepte, sin importar si hubo combinaciones que quedaron loopeando.

Ejercicio 5. Dada una MT M_1 con alfabeto $\Gamma = \{0, 1\}$:

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.

Construir una MT M_2 que ejecute M_1 sobre todas las cadenas ingresadas en forma “paralela” (al igual en el ejercicio 4), y que solo acepte si M_1 aceptó alguna de las cadenas ingresadas.

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.

No, ya que para determinar esto habría que chequear todas las entradas como en el inciso anterior, y M_1 podría loopear con algunas de ellas sin permitirnos determinar si la acepta o no.

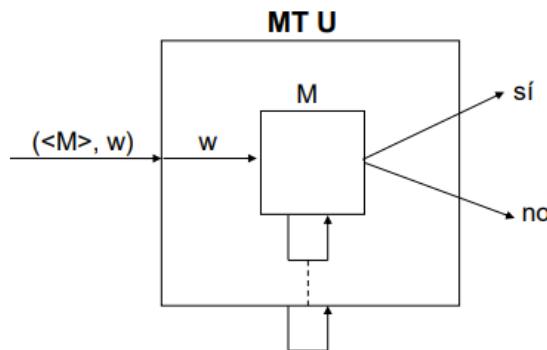
Ayuda para la parte (1): si M_1 acepta al menos una cadena, entonces existe al menos una cadena de símbolos 0 y 1, de tamaño n , tal que M_1 la acepta en k pasos. Teniendo en cuenta esto, pensar cómo M_2 podría simular M_1 considerando todas las cadenas de símbolos 0 y 1 hasta encontrar eventualmente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

PRACTICA 3 - INDECIBILIDAD

Ejercicio 1.

¿Qué es una MT universal?

Una máquina de Turing universal (MT U) es una máquina de Turing capaz de ejecutar cualquier otra MT, adquiere la noción de programa almacenado.



La MT U recibe como entrada una MT M (codificada mediante una cadena) y una cadena w, y ejecuta M a partir de w.

Ejercicio 2.

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

Números naturales pares:

1. Imprime el 0.
2. Inicializa un contador en 0.
3. Suma 2 al contador.
4. Imprime el contador.
5. Repite 3 y 4 sucesivamente.

Números enteros:

1. Inicializa un contador en 1.
2. Imprime el contador.
3. Incrementa en 1 el contador.
4. Repite 2 y 3 sucesivamente.

Números racionales:

1. Inicializo un contador X en 0.
2. Inicializo un contador Y en 1.
3. Imprimo X/Y. (0/1).
4. Incremento X en 1.
5. Imprimo X/Y. (1/1).
6. Incremento Y en 1.
7. Imprimo X/Y. ($\frac{1}{2}$).
8. Incremento Y en 1.
9. Imprimo X/Y. ($\frac{1}{3}$).
10. Incremento X en 1.
11. Si X < Y
 - 11.1. Imprimo X/Y. ($\frac{2}{3}$).
- Si X = Y
 - 11.2. Vuelvo X a 1.
12. Repite {7, 8, 9, 10, 11}.

Cadenas de Σ^* siendo $\Sigma = \{0, 1\}$:

Llevar un contador de longitud inicialmente 1.

1. Imprimir todas las cadenas de la longitud del contador en orden canónico.
2. Incrementar en 1 el contador.
3. Repetir 1 y 2.

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.

Construiría una MT M que:

1. Inicializa un contador en 0.
2. Genera una fórmula.
3. Ejecuta M_1 sobre la fórmula.
4. Si M_1 acepta:
 - 4.1. Ejecuto M_2 sobre la fórmula.
 - 4.2. Si M_2 acepta:
 - 4.2.1. Incremento el contador.
 - 4.2.2. Si El contador es igual a i :
 - 4.2.2.1. Imprimo la fórmula.
5. Repite 2, 3 y 4.

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 si $w \in L$. Construir una MT M_2 que dado w , ejecute M_1 y compare w con cada cadena de la salida de M_1 . Si encuentra una que matchee, M_2 aceptara.

Ejercicio 5.

El lenguaje $LU = \{(\langle M \rangle, w) \mid M \text{ acepta } w\}$ se conoce como lenguaje universal, y representa el problema general de aceptación. Probar que $LU \in RE$.

Ayuda: construir una MT que acepte LU .

Construir una MT M_U que acepte LU . M_U ejecutará M sobre w . Si M acepta, M_U también lo hace. Si M rechaza, M_U tambien. Y si en la ejecución M loopea, M_U tambien loopeara. Entonces, si M no acepta w puede no detenerse, por esto $LU \in RE$.

Ejercicio 6.

Una función $f : A \rightarrow B$ es total computable si existe una MT M_f que la computa para todo elemento $a \in A$. Sea la función $f_{01} : \Sigma^ \rightarrow \{0, 1\}$ tal que:*

$f_{01}(v) = 1$, si $v = (\langle M \rangle, w)$ y M para a partir de w .

$f_{01}(v) = 0$, si $v = (\langle M \rangle, w)$ y M no para a partir de w o bien $v \neq (\langle M \rangle, w)$.

Probar que f_{01} no es total computable.

Ayuda: ¿con qué problema se relaciona dicha función?

Se relaciona con el problema del halting problem.

f_{01} , al decir que responde '0' si $v = (\langle M \rangle, w)$ y M no para a partir de w o bien $v \neq (\langle M \rangle, w)$, está simulando el Halting Problem, el cual no es recursivo por lo que no es decidible.

Entonces, como para responder '0' debe decidir si M para a partir de w , podemos decir que f_{01} no podrá decidir en todos los casos, por lo que no será totalmente computable.

Ejercicio 7.

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

a. 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?

Tiene $K \cdot N_1 \cdot N_2^K$ configuraciones distintas, siendo N_1 estados, N_2 símbolos, y que recorre a lo sumo K celdas. M loopeara al repetir alguna de esas configuraciones.

Si M escribe un símbolo no blanco, lo hará en un número finito de pasos antes de repetir un estado y configuración.

Como M tiene una cinta finita utilizada hasta ese momento y un número finito de estados, hay un número finito de configuraciones posibles ($K \cdot N_1 \cdot N_2^K$).

Si supera ese número sin escribir nada, entrará en un bucle y nunca escribirá un símbolo no blanco.

Entonces, se puede probar que se puede decidir si una MT M, escribe alguna vez un símbolo no blanco a partir de la cadena vacía λ .

b. 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?

¿Qué variante introduce el hecho de tener un solo movimiento?

c. 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?

Hasta la longitud 10.

Construir una MT M que ejecute i pasos sobre cada cadena en el orden canónico, con $i \leq 10$.

1. Hacer $i := 1$.

2. Ejecutar i pasos de M sobre todas las cadenas de longitud a lo sumo i.

3. Si M acepta alguna vez, aceptar.

4. Si no, hacer $i := i + 1$ y volver al paso 2.

d. ¿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.

Si, se puede decidir este problema. Se puede construir una MT M₁ que genere todas las cadenas de a lo sumo 10 símbolos y simule sobre cada una de ellas la MT M.

1. Hacer $i := 1$.

2. Ejecutar i pasos de M sobre todas las cadenas de longitud a lo sumo 10.

3. Si M acepta alguna vez, aceptar.

4. Si no, hacer $i := i + 1$ y volver al paso 2.

Si $i > 10$, MT M₁ rechaza.

PRÁCTICA 4 - LAS REDUCCIONES

Ejercicio 1.

Considerando la reducción de HP a LU descripta en clase, responder:

a. Explicar por qué la función identidad, es decir la función que a toda cadena le asigna la misma cadena, no es una reducción de HP a LU.

La función de identidad ($f(w) = w$) no es una reducción de HP a LU ya que para que esto se cumpla debe $(\langle M \rangle, w) \in \text{HP}$ si y sólo si $f(\langle M \rangle, w) \in \text{LU}$.

Es decir, si una máquina de Turing M se detiene en la entrada w, entonces la transformación computable f debe convertir esa entrada en otra entrada que pertenezca a LU, lo que significa que una máquina debe aceptar la entrada transformada.

La diferencia está en que una máquina puede detenerse sin aceptar (HP), mientras que LU solo contiene pares donde M acepta w, no solo donde simplemente se detiene.

b. *Explicar por qué las MT M2 generadas en los pares de salida ($\langle M2 \rangle, w$), o bien paran aceptando, o bien loopean.*

Porque Mf cambia cada qR por qA, a las qA los deja en qA y cuando hay loop no hace nada, por lo que M2 solo parará en qA o loopeara.

c. *Explicar por qué la función utilizada para reducir HP a LU también sirve para reducir $HP_{(\text{complemento})}$ a $LU_{(\text{complemento})}$.*

Sirve también ya que aplica la propiedad de que si L1 se reduce a L2 entonces también $L1_{(\text{complemento})}$ se reduzca a $L2_{(\text{complemento})}$.

Esto sucede porque el complemento de HP: si ($\langle M1 \rangle, w$) no es una cadena válida, entonces el complemento de LU: ($\langle M2 \rangle, w$) tampoco es una cadena válida ($(\langle M2 \rangle, w) \notin LU$).

d. *Explicar por qué la función utilizada para reducir HP a LU no sirve para reducir LU a HP.*
Porque no aplica la propiedad de simetría. Esta propiedad solo aplica dentro de la clase R.

e. *Explicar por qué la siguiente MT Mf no computa una reducción de HP a LU: dada una cadena válida ($\langle M \rangle, w$), Mf ejecuta M sobre w, si M acepta entonces genera la salida ($\langle M \rangle, w$), y si M rechaza entonces genera la cadena 1.*

Para que Mf sea una reducción, debe cumplir:

$$(\langle M \rangle, w) \in HP \Leftrightarrow Mf(\langle M \rangle, w) \in LU$$

Pero no se cumple en todos los casos, ya que si M acepta la entrada, Mf si devuelve ($\langle M \rangle, w$), pero si M rechaza entonces genera la cadena 1.

Ejercicio 2.

Sabiendo que $LU \in RE$ y $LU_{(\text{complemento})} \in CO-RE$:

a. *Probamos en clase que existe una reducción de LU a $L\Sigma^*$. En base a esto, ¿qué se puede afirmar con respecto a la ubicación de $L\Sigma^*$ en la jerarquía de la computabilidad?*

Se puede afirmar que $L\Sigma^*$ es tan o más difícil que LU. Entonces podemos asegurar que $L\Sigma^*$ se encuentra al menos en el mismo nivel de dificultad de computabilidad que LU o más afuera.

b. *Se prueba que existe una reducción de $LU_{(\text{complemento})}$ a L. En base a esto, ¿qué se puede afirmar con respecto a la ubicación de L en la jerarquía de la computabilidad?*

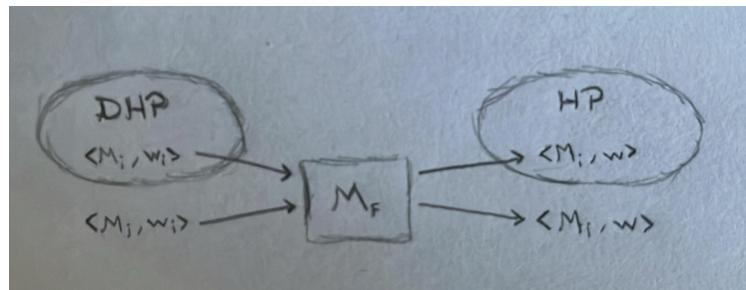
Se puede afirmar que L es igual o menos difícil que $LU_{(\text{complemento})}$. Como $LU_{(\text{complemento})} \in CO-RE$ y $L \in R$, entonces claramente L es menos difícil que $LU_{(\text{complemento})}$.

Ejercicio 3.

Sea el lenguaje $DHP = \{wi \mid Mi \text{ para a partir de } wi\}$ (considerar el orden canónico). Encontrar una reducción de DHP a HP. Comentario: hay que definir la función de reducción y probar su total computabilidad y correctitud.

Idea de la reducción: Una Mf que tome ($\langle Mi \rangle, wi$) y la convierta en ($\langle Mi \rangle, w$) donde $\{w \mid Mi \text{ para a partir de } w\}$ y w son todas las cadenas dea lo sumi i caracteres generadas en el

orden canónico. M2 ejecuta Mi sobre cada cadena de manera paralela y acepta si Mi acepta al menos una.



Reducción: M2 ejecuta Mi sobre todas las cadenas de longitud a lo sumo i en forma paralela, si Mi para (qA / qR) M2 también para.

Computabilidad: Mf copia el código de $\langle M_i \rangle$ y agrega al final del mismo todas las cadenas generadas en el orden canónico.

Correctitud:

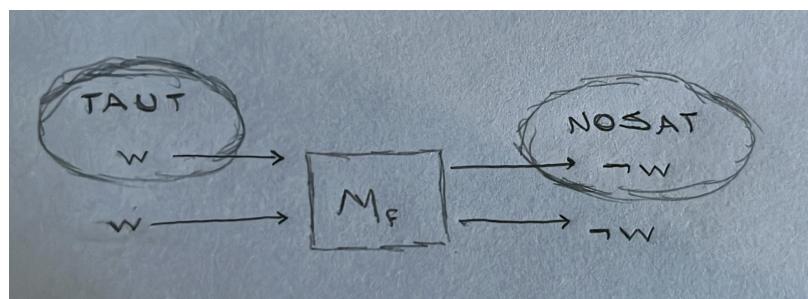
$(\langle M_i \rangle, w_i) \in DHP \rightarrow M_i \text{ para desde } w_i \rightarrow M_2 \text{ acepta o rechaza segun haya decidido } M_i \rightarrow M_2 \in DHP$.

$(\langle M_i \rangle, w_i) \notin DHP \rightarrow M_i \text{ loopea} \rightarrow M_2 \text{ loopea} \rightarrow M_2 \notin DHP$.

Ejercicio 4.

Sean TAUT y NOSAT los lenguajes de las fórmulas booleanas sin cuantificadores tautológicas (satisfacibles por todas las asignaciones de valores de verdad) e insatisfacibles (insatisfacibles por todas las asignaciones de valores de verdad). Encontrar una reducción de TAUT a NOSAT. Comentario: hay que definir la función de reducción y probar su total computabilidad y correctitud.

Idea de la reducción: negar o tomar el inverso de la respuesta de TAUT e ingresarla en NOSAT.



Reducción: dado w, la función de reducción chequea que sea una fórmula booleana, y generará

la negación de w ($\neg w$).

Computabilidad: f(w) es totalmente computable ya que solo niega w.

Correctitud:

$w \in TAUT \rightarrow M_1 \text{ acepta } w \rightarrow M_2 \text{ rechaza } w \rightarrow M_2 \notin NOSAT$.

$w \notin TAUT \rightarrow M_1 \text{ rechaza } w \rightarrow M_2 \text{ acepta } w \rightarrow M_2 \in NOSAT$.

Ejercicio 5.

Se prueba que existe una reducción de $LU_{(\text{complemento})}$ a $L\Sigma^*$ (y así, como $LU_{(\text{complemento})} \notin RE$, entonces se cumple que $L\Sigma^* \notin RE$). La reducción es la siguiente. Para toda w: $f((\langle M_1 \rangle, w)) = \langle M_2 \rangle$, tal que M_2 , a partir de su entrada v, ejecuta $|v|$ pasos de M_1 a partir de w, y acepta si M_1 no acepta. Probar que la función definida es efectivamente una reducción de $LU_{(\text{complemento})}$ a $L\Sigma^*$.

Comentario: hay que probar su total computabilidad y correctitud.

Computabilidad: $f((M_1, w))$ es totalmente computable ya que solo genera una M_2 que ejecuta v pasos de M_1 aceptando si y sólo si M_1 no acepta, es decir que rechaza o loopea, pero al ejecutar v pasos podremos decidir sobre ella.

Correctitud:

$(M_1, w) \in LU(\text{complemento}) \rightarrow M_1 \text{ acepta } w \text{ si puede decidirla en } v \text{ pasos} \rightarrow M_2 \text{ rechaza } w \rightarrow M_2 \notin L\Sigma^*$.

$\rightarrow M_1 \text{ loopea si no puede decidir } w \text{ en } v \text{ pasos} \rightarrow M_2 \text{ acepta } w \rightarrow M_2 \in L\Sigma^*$.

$(M_1, w) \notin LU(\text{complemento}) \rightarrow M_1 \text{ rechaza } w \text{ si puede decidirla en } v \text{ pasos} \rightarrow M_2 \text{ acepta } w \rightarrow M_2 \in L\Sigma^*$.

$\rightarrow M_1 \text{ loopea si no puede decidir } w \text{ en } v \text{ pasos} \rightarrow M_2 \text{ acepta } w \rightarrow M_2 \in L\Sigma^*$.

PRÁCTICA 5 - TIEMPO POLINOMIAL Y NO POLINOMIAL

Ejercicio 1.

Responder breve y claramente los siguientes incisos:

a. *¿Por qué la complejidad temporal sólo trata los lenguajes recursivos?*

Porque se ocupa de medir la complejidad de *resolver* un problema y poder medir la eficiencia de una *solución*. Por lo que los lenguajes que se encuentran por fuera de R no son del campo de la complejidad computacional por no tener una solución asegurada.

Porque para analizar el tiempo que tarda una máquina en resolver un problema, necesitamos que siempre se detenga.

b. *Probar que $n^3 = O(2^n)$.*

Ayuda: *hay que encontrar un número natural n_0 y una constante $c > 0$ tales que $n^3 \leq c \cdot 2^n$ para todo $n \geq n_0$.*

$$n^3 \leq c \cdot 2^n$$

Si $n_0 = 10$, entonces:

$$10^3 = 1000$$

$$2^{10} = 1024$$

Se cumple $1000 < 1024$

Teniendo $c = 1$: $n^3 \leq 2^n$ para todo $n \geq 10$, por lo tanto $n^3 = O(2^n)$

c. *Probar que si $T_1(n) = O(T_2(n))$, entonces $\text{TIME}(T_1(n)) \subseteq \text{TIME}(T_2(n))$.*

Ayuda: *hay que probar que si un lenguaje L está en $\text{TIME}(T_1(n))$, también está en $\text{TIME}(T_2(n))$ - recurrir directamente a la definición de orden O de una función T y de clase $\text{TIME}(T(n))$.*

Sea $L \in T_1(n)$,

L tiene una MT M que dada una entrada w de tamaño n , M hace a lo sumo $T_1(n)$.

Entonces, por definición de $O()$, L es resuelto en a lo sumo $T_1(n)$ pasos que es $\leq a \cdot c \cdot T_2(n)$ pasos, el tiempo que tarda $T_1(n)$ es \leq al tiempo que tarda $T_2(n)$ multiplicado por una constante mayor a 0,

por lo tanto $\text{TIME}(T_1(n)) \subseteq \text{TIME}(T_2(n))$.

d. ¿Cuándo un lenguaje pertenece a **P**, a **NP** y a **EXP**? ¿Por qué si un lenguaje pertenece a **P** también pertenece a **NP** y a **EXP**?

Un lenguaje pertenece a **P** si es decidable en tiempo $\text{poly}(n)$.

Un lenguaje pertenece a **NP** si es verificable en tiempo $\text{poly}(n)$.

Un lenguaje pertenece a **EXP** si es decidable en tiempo $\text{exp}(n)$.

Porque se cumple que $P \subset NP \subset EXP = P \subset NP$ se cumple ya que de no ser así, la entrada X no se utilizaría en NP y sería lo mismo que P , y $NP \subset EXP$ porque todo tiempo polinomial es menor que uno exponencial.

e. ¿Qué formula la Tesis Fuerte de Church-Turing?

Si L es decidable en tiempo $\text{poly}(n)$ por un modelo computacional físicamente realizable, también es decidable en tiempo $\text{poly}(n)$ por una MT (al menos hasta que las máquinas cuánticas sean una realidad).

f. ¿Por qué es indistinta la cantidad de cintas de las MT que utilizamos para analizar los lenguajes, en el marco de la jerarquía temporal que definimos?

Porque se ha demostrado que una MT de múltiples cintas puede ser simulada por una MT de una sola cinta con a lo sumo un aumento cuadrático en el tiempo.

Esto significa que el modelo no afecta la clase de complejidad, solo cambia por un factor polinomial, que no altera la jerarquía temporal.

g. ¿Qué codificación de cadenas se descarta en la complejidad temporal?

Se descartan codificaciones artificialmente infladas (como representaciones unarias) que aumenten artificialmente el tamaño de la entrada y distorsionen el análisis de tiempo.

Es decir aquellas entradas que me voy de tiempo con el solo hecho de leerlas.

h. ¿Por qué si un lenguaje L pertenece a **P**, también su complemento $L_{(\text{complemento})}$ pertenece a **P**?

Ayuda: hay que probar que si existe una MT M que decide L en tiempo $\text{poly}(n)$, también existe una MT M' que decide $L_{(\text{complemento})}$ en tiempo $\text{poly}(n)$.

Construimos una MT M que decide L en tiempo $\text{poly}(n)$.

Construimos una MT M' que decide $L_{(\text{complemento})}$, es decir que si M acepta M' rechaza, y si M rechaza M' acepta.

Como M decide L en tiempo $\text{poly}(n)$, M' también decidirá $L_{(\text{complemento})}$ en tiempo $\text{poly}(n)$.

i. Sea L un lenguaje de **NP**. Explicar por qué los certificados de L miden un tamaño polinomial con respecto al tamaño de las cadenas de entrada.

Porque un lenguaje está en **NP** si existe un certificado (prueba) que puede ser verificado en tiempo polinomial.

Para que la verificación sea polinomial, el certificado debe tener tamaño polinomial respecto del tamaño de la entrada.

De lo contrario, no podríamos verificarlo en tiempo polinomial.

Ejercicio 2.

Sea el lenguaje **SMALL-SAT** = $\{\varphi \mid \varphi \text{ es una fórmula booleana sin cuantificadores en la forma normal conjuntiva (o FNC), y existe una asignación de valores de verdad que la}$

satisface en la que hay a lo sumo 3 variables con valor de verdad verdadero}. Probar que $\text{SMALL-SAT} \in P$.

Comentario: las fórmulas φ en la forma FNC son conjunciones de disyunciones de variables o variables negadas; p.ej. $(x_1 \vee x_2) \wedge x_4 \wedge (\neg x_3 \vee x_5 \vee x_6)$.

Ayuda: una MT que decide SMALL-SAT debe contemplar asignaciones con cero, uno, dos y hasta tres valores de verdad verdadero.

Construir una MT M que emplee una tabla de verdad donde solo aparezcan aquellas entradas que tienen a lo sumo 3 variables con valor verdadero.

EL conjunto de asignaciones a chequear para n variables son

$$\sum_{k=0}^3$$

$$(n \text{ tomados de } a k) = (n \text{ tom. de } a 0) + (n \text{ tom. de } a 1) + (n \text{ tom. de } a 2) + (n \text{ tom. de } a 3)$$

Este número es polinómico = $O(n^3)$

Entonces, el tiempo total es $O(n^3 \cdot |\varphi|)$ = polinómico $\in P$.

Ejercicio 3.

Dados los dos lenguajes siguientes, (1) justificar por qué no estarían en P , (2) probar que están en NP , (3) justificar por qué sus complementos no estarían en NP :

a. El problema del conjunto dominante de un grafo consiste en determinar si un grafo no dirigido tiene un conjunto dominante de vértices. Un subconjunto D de vértices de un grafo G es un conjunto dominante de G , si todo vértice de G fuera de D es adyacente a algún vértice de D . El lenguaje que representa el problema es $\text{DOM-SET} = \{(G, K) \mid G \text{ es un grafo no dirigido y tiene un conjunto dominante de } K \text{ vértices}\}$.

(1) DOM-SET no estaría en P ya que requiere comprobar que cada vértice del grafo que no pertenezca a D sea adyacente a alguno de los k vértices de D . Lo que implica un tiempo de $O(n^k)$ = EXP.

(2) DOM-SET se puede verificar en tiempo poly(n) a partir de un certificado de tamaño polinómico por lo que está en NP . Nos dan un certificado con una posible solución y podemos verificar que ese certificado es verdadero en tiempo poly(n).

(3) DOM-SET(complemento) sería comprobar que G no tiene ningún grafo dominante. Para esto recibimos un certificado con todos los posibles subgrafos y deberíamos comprobar que ninguna de todas las posibles combinaciones cumple que todo el resto de los vértices son adyacentes a alguno de ellos. Solo leer este certificado nos llevaría tiempo EXP(n) por lo que no estaría en NP .

b. El problema de los grafos isomorfos consiste en determinar si dos grafos son isomorfos. Dos grafos son isomorfos si son idénticos salvo por la denominación de sus arcos. P.ej., dado el grafo $G_1 = (\{1, 2, 3, 4\}, \{(1, 2), (2, 3), (3, 4), (4, 1)\})$, el grafo $G_2 = (\{1, 2, 3, 4\}, \{(1, 2), (2, 4), (4, 3), (3, 1)\})$ es isomorfo a G_1 . El lenguaje que representa el problema de los grafos isomorfos es $\text{ISO} = \{(G_1, G_2) \mid G_1 \text{ y } G_2 \text{ son grafos isomorfos}\}$.

(1) ISO no estaría en P porque comprobar que el grafo G_2 es isomorfo a G_1 requeriría tomar los vértices de G_2 y probar todas las posibles combinaciones para ver si alguna de ellas es isomorfa a G_1 .

(2) ISO estaría en NP ya que si recibimos un certificado con una solución podemos comprobarla en tiempo poly(n).

(3) ISO_(complemento) sería comprobar que ninguna combinación de G2 es isomorfa a G1. Para comprobar que está en NP deberíamos poder verificar un certificado en tiempo poly(n). El certificado traería todas las posibles combinaciones de los vértices de G2 y habría que comprobarlas una por una, por lo que nos tomaría tiempo EXP(n) el solo hecho de leerlas (n^n).

Ejercicio 4.

Se prueba que $NP \subseteq EXP$. La prueba es la siguiente. Si $L \in NP$, entonces existe una MT M que, para toda cadena de entrada w, verifica en tiempo $poly(|w|)$ si $w \in L$, con la ayuda de un certificado x tal que $|x| \leq p(|w|)$ - p es un polinomio -, y de esta manera, se puede construir una MT M' que decida en tiempo $exp(|w|)$ si $w \in L$, sin usar ninguna cadena adicional: M' simplemente barre todos y cada uno de los certificados posibles x de w. Se pide explicar por qué M' efectivamente tarda tiempo $exp(|w|)$.

Ayuda: como $|x| \leq p(|w|)$ y los símbolos de x pertenecen a un alfabeto de k símbolos, ¿cuántos certificados x de tamaño $p(|w|)$ tiene a lo sumo una cadena w?

Una cadena w tiene a lo sumo $k^{|w|}$ certificados x.

Entonces M' barre $k^{|w|}$ certificados de x para cada cadena w de entrada = $O(w.k^{|w|})$. Por lo tanto M' tiene tarda tiempo $exp(|w|)$.

PRACTICA 6 - PERTENENCIA Y REDUCCIONES

Ejercicio 1.

Responder breve y claramente:

a. Dar un esquema de prueba de la transitividad de las reducciones polinomiales.

Ayuda: en clase hicimos lo propio con las reducciones generales.

Si tenemos 2 lenguajes A y B, podemos decir que A se reduce polinomialmente a B si existe una función f_1 que transforma instancias del problema A a instancias del problema B en tiempo polinomial.

Entonces, si $x \in A$, $f_1(x) \in B$.

Luego, si tenemos el lenguaje C, podemos decir que B se reduce polinomialmente a C si existe una función f_2 que transforma instancias del problema B en instancias del problema C en tiempo polinomial.

Entonces, si $y \in B$, $f_2(y) \in C$.

Por esto, podemos probar la transitividad de las reducciones polinomiales demostrando que si $A \leq_p B$, y $B \leq_p C$, entonces $A \leq_p C$.

Porque:

Si $x \in A \Rightarrow f_1(f_2(x)) \in C$.

Si $x \notin A \Rightarrow f_1(f_2(x)) \notin C$.

$f_1(f_2(x))$ se computa en tiempo polinomial ya que f_1 lo hace y f_2 también. Entonces el tiempo es $poly(poly(n))$ que sigue siendo $poly(n)$.

b. ¿Cuándo un lenguaje es NP-difícil y cuándo es NP-completo?

. Un lenguaje L es NP-difícil si todos los problemas de NP se reducen en tiempo polinomial a L.

Es decir, todo $L_i \in NP$ cumple $L_i \leq_p L$.

. Un lenguaje L es NP-completo si cumple 2 condiciones:

$L \in NP$.

L es NP-difícil.

Entonces NP-completo = $NP \cap NP\text{-difícil}$.

c. ¿Por qué si $P \neq NP$, un lenguaje NP-completo no pertenece a P?

Porque si $L \in NP$ -completo entonces también es NP-difícil, es decir que todo $L_i \in NP$ se reduce a L. Por lo que si L perteneciera a P, entonces estaríamos diciendo que todo $L_i \in NP$ pertenece a P, y no porque $P \neq NP$. Absurdo.

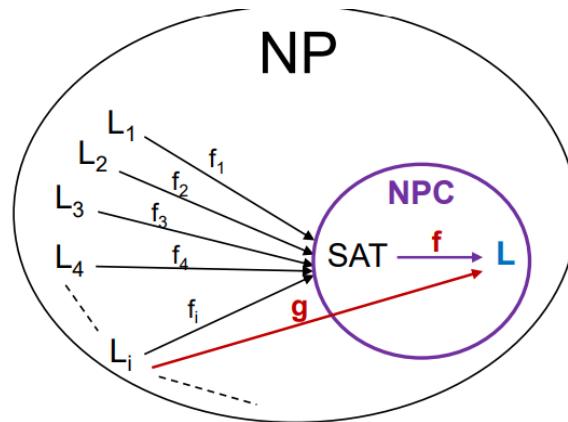
d. Enunciar el esquema visto en clase para agregar un lenguaje a la clase NPC.

Para agregar un lenguaje L a NPC se debe:

1. Probar que L pertenece a NP.
2. Construir una reducción polinomial f de SAT a L

Se cumple que L es NP-completo:

- Sea g la composición de f_i con f
- La función g es una reducción polinomial de L_i a L (las reducciones polinomiales son transitivas)
- Como lo anterior vale para todo L_i de NP, entonces L es NP-completo.



e. ¿Cuándo se sospecha que un lenguaje de NP está en NPI?

Cuando un lenguaje:

- No cuentan con una MT de tiempo polinomial que los decida.
- No cuentan con una reducción polinomial desde un lenguaje NP-completo.

Ejercicio 2.

Ayuda: recurrir directamente a la definición de NPC.

Probar:

a. Si $L_1 \in NPC$ y $L_2 \in NPC$, entonces $L_1 \leq_p L_2$ y $L_2 \leq_p L_1$.

Sean $L_1 \in NPC$ y $L_2 \in NPC$,

ambos L_1 y $L_2 \in NP\text{-difícil}$ y $\in NP$,

por lo tanto L_1 que es NP se reduce en tiempo polinomial a L_2 que es NP-difícil y viceversa por:

“Un lenguaje L es NP-difícil si todos los problemas de NP se reducen en tiempo polinomial a L”

b. Si $L_1 \leq_p L_2, L_2 \leq_p L_1$, y $L_1 \in NPC$, entonces $L_2 \in NPC$.

Si $L_1 \in NPC$, entonces también $\in NP$, y si se reduce en tiempo polinomial a L_2 entonces **L2 es NP-difícil**.

Si $L_1 \in NPC$, entonces también $\in NP$ -difícil, y si L_2 se reduce en tiempo polinomial a L_1 entonces **L2 es NP**.

Entonces L_2 cumple:

$\in NP$.

$\in NP$ -difícil.

Entonces $L_2 \in NPC$.

Ejercicio 3.

Un lenguaje es CO-NP-completo si todos los lenguajes de CO-NP se reducen polinomialmente a él. Probar que SAT^c es CO-NP-completo.

Ayuda: $L_1 \leq L_2$ si $L_1^c \leq L_2^c$.

Sea SAT^c podemos construir una función f que reduzca SAT^c a L^c en tiempo polinomial.

Para probar que SAT^c es CO-NP-completo debemos lograr que todo $L_i^c \in CO-NP$ se pueda reducir polinomialmente a SAT^c mediante una función f_i .

Si todo $L_i \in NP$ se reduce polinomialmente a SAT y SAT se puede reducir polinomialmente a L,

Entonces todo $L_i^c \in CO-NP$ se reduce polinomialmente a L^c y se cumple que SAT^c es CO-NP-completo.

Se hace la misma demostración que para los NPC pero con los complementos ya que $L_1 \leq L_2$ si $L_1^c \leq L_2^c$.

Ejercicio 4.

Sean los lenguajes A y B, tales que $A \neq \emptyset$, $A \neq \Sigma^$, y $B \in P$. Probar: $(A \cap B) \leq_p A$.*

Ayuda: intentar con una reducción polinomial que, dada una cadena w, lo primero que haga sea chequear si $w \in B$, teniendo en cuenta que existe un elemento e que no está en A.

Podemos proponer una reducción mediante una M_f que chequea que $w \in B$ en tiempo polinómico ya que $B \in P$, si $w \in B$ envía la entrada así tal cual, ya que si también existe en A quedará dentro de A por $A \cap B$, y si no existe en A quedará fuera de A por no existir en dicho conjunto simplemente.

Si $w \notin B$ se envía fuera de A porque no pertenece a $A \cap B$.

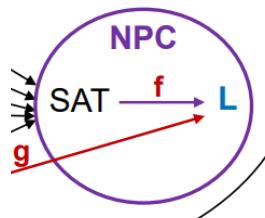
Esta reducción se realiza en tiempo polinómico ya que $B \in P$ y solo debe ser chequeado B.

Ejercicio 5.

Sea el lenguaje SH-s-t = $\{(G, s, t) \mid G \text{ es un grafo no dirigido y tiene un camino de Hamilton del vértice } s \text{ al vértice } t\}$. Un grafo $G = (V, E)$ tiene un camino de Hamilton del vértice s al vértice t si G tiene un camino entre s y t que recorre todos los vértices restantes una sola

vez. Probar que SH-s-t es NP-completo. Ayuda: se sabe que CH, el lenguaje correspondiente al problema del circuito hamiltoniano, es NP-completo.

Planteamos una reducción del Circuito de Hamilton (CH) al Camino de Hamilton (SH-s-t), como si CH fuera SAT y SH-s-t fuera L:



PRACTICA 7 - JERARQUÍA ESPACIO-TEMPORAL

Ejercicio 1.

Responder breve y claramente:

a. *¿Por qué en la complejidad espacial se utilizan MT con una cinta de entrada de sólo lectura?*

Porque de esta manera podemos lograr un espacio menor que lineal, es decir menor que $O(n)$, ya que la cadena de entrada mide n , y si la contamos como espacio ocupado nunca conseguiremos un orden menor que lineal.

b. *¿Por qué si una MT tarda tiempo $\text{poly}(n)$ entonces ocupa espacio $\text{poly}(n)$, y si ocupa espacio $\text{poly}(n)$ puede llegar a tardar tiempo $\text{exp}(n)$?*

Porque el espacio siempre es menor o igual que el tiempo.

Ej: Si una máquina tarda como máximo 5 pasos, ¿cuantas celdas puede recorrer como máximo? 5.

Y si ocupa tiempo $\text{poly}(n)$ puede llegar a tardar tiempo $\text{exp}(n)$ porque si algo trabaja en un espacio que es cualquier función, tardará un tiempo elevado a esa función, entonces tendremos un tiempo polinomial elevado a un espacio polinomial, y como queda n elevado a la n termina siendo tiempo exponencial.

c. *¿Por qué los lenguajes de la clase LOGSPACE son tratables?*

Porque “Si algo trabaja en un espacio que es cualquier función, entonces trabaja en un tiempo elevado a esa función.”

Si el espacio en el que trabajamos es logarítmico, la función es logarítmica.

Entonces tengo una función polinomial elevada a una función logarítmica, $n^{\log_2 n}$, que se simplifica a $2^{\log_2 n}$ que es igual a n , por lo que se acota a $O(n)$.

Ejercicio 2.

Describir la idea general de una MT M que decida el lenguaje $L = \{a^n b^n \mid n \geq 1\}$ en espacio logarítmico.

Ayuda: basarse en el ejemplo mostrado en clase.

Planteamos una MT M que utilizará 4 cintas además de la que está la entrada. La cinta de entrada solo será leída.

1. En la cinta 2 llevaremos un contador i que iniciará en 1.

2. En la cinta 3 llevaremos un contador j que iniciara en n , si n es impar rechaza.
3. En la cinta 4 se copia el símbolo i de w .
4. En la cinta 5 se copia el símbolo j de w .
5. Si $i > j$: si los símbolos son b en la cinta 4 y a en la cinta 5, acepta, sino rechaza.
Si $i < j$: si los símbolos no son a en la cinta 4 y b en la cinta 5, rechaza.
6. Hace $i = i + 1$.
7. Hace $j = j - 1$.
8. Vuelve al paso 3.

Ejercicio 3.

Describir la idea general de una MT M que decida el lenguaje SAT en espacio polinomial.

Ayuda: la generación y la evaluación de una asignación de valores de verdad se pueden efectuar en tiempo polinomial.

Planteamos una MT M que dada una entrada w que la recibe en la cinta 1:

1. La copia en la cinta 2 \rightarrow espacio $\text{poly}(n)$ porque de no ser así la entrada no podría ser leída.
2. Genera una asignación de valores de verdad en la cinta 3 \rightarrow espacio $\text{poly}(n)$ porque si tarda tiempo $\text{poly}(n)$ en generarla, la asignación entonces ocupará espacio $\text{poly}(n)$ o menos.
3. Evalúa la asignación de la cinta 3 sobre la entrada de la cinta 2 \rightarrow espacio $\text{poly}(n)$ porque la evaluación se realiza sobre el espacio ya ocupado por lo que no suma nuevo espacio ocupado.
4. Vuelve al paso 1.

El espacio entonces el $\text{poly}(n)$ porque se ocupa $\text{poly}(n)$ en la cinta 2 y $\text{poly}(n)$ en la cinta 3
 $\Rightarrow \text{poly}(n) \cdot \text{poly}(n) = \text{poly}(n)$.

Ejercicio 4.

Justificar por qué el lenguaje QSAT no pertenecería a P ni a NP.

Ayuda: ¿qué forma tienen los certificados asociados a QSAT?

El lenguaje QSAT no pertenece a P ni a NP porque para pertenecer a NP debe poder ser verificable en tiempo $\text{poly}(n)$ mediante un certificado sucinto, y para este problema el certificado deja de ser sucinto y pasa a ser un árbol con todas las posibles combinaciones, las cuales ocupan espacio exponencial.

Y no está en P porque debería poder ser decidible en tiempo $\text{poly}(n)$ y para eso deberíamos probar todas las posibles combinaciones que ocupan espacio exponencial, por lo que el tiempo es igual o mayor a exponencial.

(Fuerza Bruta).

Ejercicio 5.

Probar que $NP \subseteq \text{PSPACE}$.

Ayuda: Si L pertenece a NP, entonces existe una MT M_1 capaz de verificar en tiempo $\text{poly}(|w|)$ si una cadena w pertenece a L , con la ayuda de un certificado x de tamaño $\text{poly}(|w|)$. De esta manera, existe también una MT M_2 que decide L en espacio $\text{poly}(|w|)$, sin la ayuda de ningún certificado.

PSPACE es la clase de los lenguajes decidibles en espacio $\text{poly}(n)$.

Y para un lenguaje NP puedo tener una MT que lo verifique con la ayuda de un certificado que ocupa espacio $\text{poly}(n)$.

Entonces, para decidir un lenguaje NP en espacio poly(n) puedo generar todos los posibles certificados (que cada uno ocupa espacio poly(n)) y verificarlos con la MT, si esta acepta yo acepto y si rechaza yo rechazo.

Pero todos estos certificados no puedo generarlos a la vez porque eso ocuparía espacio exponencial, por lo que debería generar uno, verificarlo, y luego reutilizar ese mismo espacio para generar otro, y así sucesivamente.

Entonces, si reutilizo el espacio de cada certificado que es espacio polinomial, estaría ocupando espacio polinomial y decidiendo un lenguaje NP, por lo que puedo decir que $NP \subseteq PSPACE$.

Ejercicio 6.

Supongamos que existe una MT M de tiempo polinomial que, dado un grafo G , devuelve un circuito de Hamilton de G si existe o responde no si no existe. Describir la idea general de una MT M' que, utilizando M , decida en tiempo polinomial si un grafo G tiene un circuito de Hamilton.

Ayuda: basarse en el ejemplo mostrado en clase con FSAT y SAT.

A partir de M' puedo construir M tal que si M' resuelve FCH en tiempo polinomial, M decide CH en tiempo polinomial.

Idea general:

1. Ingresa una entrada w a M .
2. M ejecuta M' sobre w .
3. M toma la respuesta de M' :
 - 3.1. Si M' devuelve un Circuito de Hamilton $\rightarrow M$ acepta.
 - 3.2. Si M' responde no $\rightarrow M$ rechaza.

Ejercicio 7.

Sea la MTP M que definimos en clase para decidir probabilísticamente el lenguaje MAYSAT = $\{\varphi \mid \varphi \text{ es una fórmula booleana satisfactible por más de la mitad de las posibles asignaciones de verdad}\}$. Hemos indicado que para toda φ , si $\varphi \in MAYSAT$ entonces M la acepta con probabilidad $> 1/2$, y si $\varphi \notin MAYSAT$ entonces M la rechaza con probabilidad $\geq 1/2$. Precisando más la primera probabilidad: asumiendo que M tarda $p(n)$, si $\varphi \in MAYSAT$ entonces M la acepta con probabilidad $\geq 1/2 + 1/2^{p(n)}$. Explicar por qué.

Ayuda: en tiempo $p(n)$, M puede producir $2^{p(n)}$ computaciones posibles, y entonces, ¿cuántas son de aceptación como mínimo si $\varphi \in MAYSAT$?

Si M puede producir $2^{p(n)}$ computaciones posibles, entonces si $\varphi \in MAYSAT$, al menos $2^{p(n)-1} + 1$ son de aceptación.

Entonces si M en cada paso elige aleatoriamente una entre dos continuaciones, cada una con probabilidad $1/2$, y hay $2^{p(n)}$ continuaciones, de las cuales $2^{p(n)-1} + 1$ son de aceptación si $\varphi \in MAYSAT$,

la probabilidad de que M acepte será $1/2$ debido a la aleatoriedad en la elección de continuidad $+ 1/2^{p(n)}$ que es un número que cuanto más grande sea $p(n)$ más cerca de 0 estará, por lo que tiende a 0.

Entonces, la probabilidad de encontrar una combinación sobre la mitad + 1 posibles, es la mitad + algo que tiende a 0.

Ejercicio 8.

Considerando el ejemplo de computación cuántica mostrado en clase, indicar los resultados posibles cuando en lugar de arrancar con el estado inicial $|00\rangle$, la computación arranca con:

a. El estado inicial $|01\rangle$.

1. Sea un registro cuántico de dos cubits en el estado $|01\rangle$.
2. Aplicando sobre el primer cúbbit la puerta de Hadamard, se obtiene la superposición de los estados $|01\rangle$ y $|11\rangle$.
3. Aplicando sobre los dos cubits la puerta CNOT, se obtiene la superposición de los estados $|01\rangle$ y $|11\rangle$.
4. Finalmente, leyendo el registro, se obtiene el estado $|01\rangle$ y $|11\rangle$, cada uno con probabilidad $1/2$, y se destruye la superposición.

b. El estado inicial $|10\rangle$.

1. Sea un registro cuántico de dos cubits en el estado $|10\rangle$.
2. Aplicando sobre el primer cúbbit la puerta de Hadamard, se obtiene la superposición de los estados $|10\rangle$ y $|00\rangle$.
3. Aplicando sobre los dos cubits la puerta CNOT, se obtiene la superposición de los estados $|11\rangle$ y $|00\rangle$.
4. Finalmente, leyendo el registro, se obtiene el estado $|11\rangle$ y $|00\rangle$, cada uno con probabilidad $1/2$, y se destruye la superposición.

c. El estado inicial $|11\rangle$.

1. Sea un registro cuántico de dos cubits en el estado $|11\rangle$.
2. Aplicando sobre el primer cúbbit la puerta de Hadamard, se obtiene la superposición de los estados $|11\rangle$ y $|01\rangle$.
3. Aplicando sobre los dos cubits la puerta CNOT, se obtiene la superposición de los estados $|10\rangle$ y $|01\rangle$.
4. Finalmente, leyendo el registro, se obtiene el estado $|10\rangle$ y $|01\rangle$, cada uno con probabilidad $1/2$, y se destruye la superposición.

PRACTICA 8 - LÓGICA PROPOSICIONAL

Ejercicio 1.

Dada la siguiente información:

“Si el unicornio es mítico, entonces es inmortal, pero si no es mítico, entonces es un mamífero mortal. Si el unicornio es o inmortal o un mamífero, entonces tiene un cuerno. El unicornio es mágico sí tiene un cuerno”.

Simbolizar en el Cálculo de Enunciados y responder:

p: “El unicornio es mítico”

$\neg p$: “El unicornio no es mítico”

q: “El unicornio es mortal”

$\neg q$: “El unicornio es inmortal”

r: “El unicornio tiene un cuerno”

$\neg r$: “El unicornio no tiene un cuerno”

s: “El unicornio es mágico”

$\neg s$: “El unicornio no es mágico”

t: “El unicornio es un mamífero”

$\neg t$: “El unicornio no es un mamífero”

$$(p \rightarrow \neg q) , (\neg p \rightarrow (t \wedge q)) . ((\neg q \vee t) \rightarrow r) . (r \rightarrow s) .$$

“Si el unicornio es mítico, entonces es inmortal, entonces tiene un cuerno, entonces es mágico”

“Si el unicornio no es mítico, entonces es un mamífero mortal, entonces tiene un cuerno, entonces es mágico”

I. El unicornio es mítico?. Fundamentar.

Conclusión A: “Si el unicornio es mítico, entonces es inmortal, entonces tiene un cuerno, entonces es mágico”

$(p \rightarrow \neg q)$	$(\neg p \rightarrow (t \wedge q))$	$((\neg q \vee t) \rightarrow r)$	$(r \rightarrow s)$	$(p \rightarrow (\neg q \rightarrow (r \rightarrow s)))$
V V VF	FV V F F F	FV F F V V	V V V	V V VF V V V V

El unicornio es mítico, ya que la argumentación $A_1, A_2, \dots, A_n \therefore A$ es válida, con:

$$A_1, A_2, \dots, A_n = (p \rightarrow \neg q) , (\neg p \rightarrow (t \wedge q)) . ((\neg q \vee t) \rightarrow r) . (r \rightarrow s).$$

$$A = (p \rightarrow (\neg q \rightarrow (r \rightarrow s))).$$

II. El unicornio no es mítico?. Fundamentar.

Conclusión A: “Si el unicornio no es mítico, entonces es un mamífero mortal, entonces tiene un cuerno, entonces es mágico”

$$A = (\neg p \rightarrow ((t \wedge q) \rightarrow (r \rightarrow s)))$$

$(p \rightarrow \neg q)$	$(\neg p \rightarrow (t \wedge q))$	$((\neg q \vee t) \rightarrow r)$	$(r \rightarrow s)$	$(\neg p \rightarrow ((t \wedge q) \rightarrow (r \rightarrow s)))$
V V VF	FV V F F F	FV F F V V	V V V	FV V F F F V V V V

El unicornio no es mítico ya que la argumentación $A_1, A_2, \dots, A_n \therefore A$ es válida, con:

$$A_1, A_2, \dots, A_n = (p \rightarrow \neg q) , (\neg p \rightarrow (t \wedge q)) . ((\neg q \vee t) \rightarrow r) . (r \rightarrow s).$$

$$A = (\neg p \rightarrow ((t \wedge q) \rightarrow (r \rightarrow s))).$$

III. El unicornio es mágico?. Fundamentar.

Conclusión A: “El unicornio es mágico si tiene un cuerno. Tiene un cuerno si es inmortal o es mamífero. Es inmortal si es mítico. Es mamífero si no es mítico.”

$$A = ($$

$(p \rightarrow \neg q)$	$(\neg p \rightarrow (t \wedge q))$	$((\neg q \vee t) \rightarrow r)$	$(r \rightarrow s)$	$(\neg p \rightarrow ((t \wedge q) \rightarrow (r \rightarrow s)))$
V V VF	FV V F F F	FV F F V V	V V V	FV V F F F V V V V

Sabiendo ahora que el unicornio es inmortal, responder:

IV. El unicornio no es mágico?. Fundamentar.

Conclusión A: "El unicornio es inmortal, entonces es mítico y tiene un cuerno, entonces es mágico".

$$A = (\neg q \rightarrow ((p \wedge r) \rightarrow s))$$

$(p \rightarrow \neg q)$	$(\neg p \rightarrow (t \wedge q))$	$((\neg q \vee t) \rightarrow r)$	$(r \rightarrow s)$	$(\neg q \rightarrow ((p \wedge r) \rightarrow s))$
V V VF	FV V F F F	FV F F V V	V V V	VF <u>V</u> V V V <u>V</u> V

Ayuda: simbolizar como forma argumentativa y determinar si es válida o no.

Ver def. 1.28, Hamilton.

Ejercicio 2.

Para cada una de las siguientes argumentaciones, escribir una forma argumentativa que se corresponda con ella y determinar si es válida o inválida.

I. Si la función f no es continua, entonces la función g no es diferenciable. g es diferenciable.

Por lo tanto, f no es continua.

p: "f es continua"

q: "g es diferenciable"

$$A_1, A_2, \dots, A_n: (\neg p \rightarrow \neg q) \cdot q$$

Conclusión A: $\neg p$

$(\neg p \rightarrow \neg q)$	q	$\neg p$
FV V FV	V	FV

La argumentación es inválida ya que es posible asignar valores de verdad a las variables de enunciado de modo tal que las premisas tomen el valor V y la conclusión el valor F.

II. Si Juan ha instalado la calefacción central, entonces ha vendido su coche o ha pedido dinero prestado al banco. Juan no ha vendido su coche ni ha pedido dinero prestado al banco. Por lo tanto, Juan no ha instalado la calefacción central.

p: "Juan ha instalado la calefacción central"

q: "Juan ha vendido su coche"

r: "Juan ha pedido dinero prestado al banco"

$$A_1, A_2, \dots, A_n: (p \rightarrow (q \vee r)) \cdot (\neg q \wedge \neg r)$$

Conclusión A: $\neg p$

$(p \rightarrow (q \vee r))$	$(\neg q \wedge \neg r)$	$\neg p$
F V F F F	VF V VF	VF

La asignación es válida porque todas las posibles asignaciones de valores de verdad de las variables de modo tal que todas las premisas toman el valor V, la conclusión también toma el valor V.

Ayuda: Ver def. 1.28, Hamilton.

Ejercicio 3.

Una fórmula booleana es satisfactible si existe al menos una combinación de valores de verdad (una asignación de verdadero o falso a las variables) que hace que la fórmula sea verdadera. Dados los siguientes enunciados, determinar cuales son verdaderos y cuáles son falsos. Justificar en cada caso.

I. Una fórmula que es una tautología es satisfactible.

VERDADERO. Como una fórmula que es una tautología es satisfacible por todas las combinaciones de valores de verdad posibles, cumple que al menos una la satisface por lo que también es satisfactible.

II. Una fórmula que es satisfactible es una tautología.

FALSO. No siempre se cumple, ya que una fórmula que es satisfactible tiene al menos una combinación de valores de verdad que la satisface, pero no se asegura que todas la satisfacen.

III. Una fórmula que no es satisfactible es una contradicción.

VERDADERO. Que no sea satisfacible significa que no hay ni siquiera una combinación que la satisface, por lo que es una contradicción.

IV. Una fórmula que es una contradicción no puede ser satisfactible.

VERDADERO. Si es una contradicción no existe ni siquiera una combinación de valores de verdad que la satisface.

Ayuda: Ver def. 1.5, Hamilton.

Ejercicio 4.

Un conjunto de fórmulas es satisfactible si existe al menos una asignación de valores de verdad (una interpretación) que hace que todas las fórmulas del conjunto sean verdaderas al mismo tiempo. Determinar si los siguientes conjuntos son satisfactibles:

I. $\{p, (p \rightarrow q), r\}$

p	$(p \rightarrow q)$	r
V	V V V	V

Es satisfactible.

II. $\{p, q, (p \wedge \neg p)\}$

p	q	$(p \wedge \neg p)$
---	---	---------------------

V	V	V F FV
----------	----------	---------------

No es satisfactible. Nunca una variable y su negación podrán ser verdaderas.

Ejercicio 5.

Sean A, B fbf que cumplen que $(\neg A \vee B)$ es tautología. Sea C una fbf cualquiera.

Determinar, si es posible, cuáles de las siguientes fbf son tautologías y cuales contradicciones. Justificar las respuestas.

I. $((\neg(A \rightarrow B)) \rightarrow C)$

Cuando $B = V \rightarrow A$ puede ser V o F

Cuando $B = F \rightarrow A = F$ porque $\neg A$ tiene que ser V

$(A \rightarrow B)$ también es tautología.

Por lo que $(\neg(A \rightarrow B)) = F$

Por lo que tanto si $C = F$ o si $C = V$, $((\neg(A \rightarrow B)) \rightarrow C) = V$

Por lo tanto es **tautología**.

II. $(C \rightarrow ((\neg A) \vee B))$

$((\neg A) \vee B)$ es siempre V porque es una tautología

Por lo que tanto si $C = F$ o si $C = V$, $(C \rightarrow ((\neg A) \vee B)) = V$

Por lo tanto es **tautología**.

III. $((\neg A) \rightarrow B)$

Cuando $B = V \rightarrow A$ puede ser V o F

Cuando $B = F \rightarrow A = F$ porque $\neg A$ tiene que ser V

Por lo tanto cuando $B = V$, $((\neg A) \rightarrow B)$ siempre va a ser V independientemente del valor de $\neg A$

Por lo tanto cuando $B = F$, $\neg A = V$ entonces $((\neg A) \rightarrow B) = F$

Por lo tanto **no es tautología ni contradicción**.

Ayuda: Ver def. 1.5, Hamilton.

Ejercicio 6.

Responer y justificar:

I. ¿“ $(p \rightarrow q)$ ” es lógicamente equivalente a “ $(p \vee \neg q)$ ”?

p	\rightarrow	q	\leftrightarrow	p	v	\neg	q
V	V	V	V	V	V	F	V
V	F	F	F	V	V	V	F
F	V	V	F	F	F	F	V
F	V	F	V	F	V	V	F

“(p → q)” no es lógicamente equivalente a “(p ∨ ¬q)” ya que la forma enunciativa “(p → q)” ↔ “(p ∨ ¬q)” NO es una tautología.

II. ¿“ $(p \leftrightarrow q)$ ” es lógicamente equivalente a “ $((p \rightarrow q) \wedge (q \rightarrow p))$ ”?

p	\leftrightarrow	q	\leftrightarrow	(p)	\rightarrow	q)	\wedge	(q	\rightarrow	p)
V	V	V	V	V	V	V	V	V	V	V
V	F	F	V	V	F	F	F	F	V	V
F	V	V	F	F	V	V	F	V	F	F
F	V	F	V	F	V	F	V	F	V	F

“(p ↔ q)” no es lógicamente equivalente a “((p → q) ∧ (q → p))” ya que la forma enunciativa “(p ↔ q)” ↔ “((p → q) ∧ (q → p))” NO es una tautología.

III. ¿“(¬(p ∧ q))” es lógicamente equivalente a “(¬p ∨ ¬q)” ?

(¬	(p	\wedge	q)	\leftrightarrow	(¬	p	v	\neg	q)
F	V	V	V	V	F	V	F	F	V
V	V	F	F	V	F	V	V	V	F
V	F	F	V	V	V	F	V	F	V
V	F	F	F	V	V	F	V	V	F

“(¬(p ∧ q))” es lógicamente equivalente a “(¬p ∨ ¬q)” ya que la forma enunciativa “(¬(p ∧ q))” ↔ “(¬p ∨ ¬q)” es una tautología.

IV. ¿“(¬(p ∨ q))” es lógicamente equivalente a “(p ∧ q)” ?

(¬	(p	v	q)	\leftrightarrow	(p	\wedge	q)
F	V	V	V	F	V	V	V
F	V	V	F	V	V	F	F
F	F	V	V	V	F	F	V
V	F	F	F	F	F	F	F

“(¬(p ∨ q))” no es lógicamente equivalente a “(p ∧ q)” ya que la forma enunciativa “(¬(p ∨ q))” ↔ “(p ∧ q)” NO es una tautología.

Ayuda: ver def. 1.7, Hamilton.

Ejercicio 7.

Sea # el operador binario definido como $p \# q = (p \wedge \neg q) \vee (\neg p \wedge q)$.

I. Probar que # es asociativo, es decir, $x \# (y \# z)$ es lógicamente equivalente a $(x \# y) \# z$.

$p \# q = (p \wedge \neg q) \vee (\neg p \wedge q) \Rightarrow$ quiere decir que $p \# q$ es verdadero si solo uno de p o q es verdadero.

$$((x \wedge \neg(y \wedge \neg z) \vee (\neg y \wedge z)) \vee (\neg x \wedge ((y \wedge \neg z) \vee (\neg y \wedge z))) \leftrightarrow (((x \wedge \neg y) \vee (\neg x \wedge y)) \wedge \neg z) \vee (\neg((x \wedge \neg y) \vee (\neg x \wedge y)) \wedge z))$$

x	y	z	y#z	x#y	x#(y#z)	\leftrightarrow	(x#y)#z
V	V	V	F	F	V	V	V
V	V	F	V	F	F	F	F
V	F	V	V	V	F	F	F
V	F	F	F	V	V	V	V
F	V	V	F	V	F	F	F
F	V	F	V	V	V	V	V
F	F	V	V	F	V	V	V
F	F	F	F	F	F	F	F

Se cumple que $x\#(y\#z) \leftrightarrow (x\#y)\#z$ es una tautología, por lo que son lógicamente equivalentes, por lo que el operador # es asociativo.

II. Probar que # es conmutativo, es decir, $y\#z$ es lógicamente equivalente a $z\#y$.

y	z	y#z	\leftrightarrow	z#y
V	V	F	V	F
V	F	V	V	V
F	V	V	V	V
F	F	F	V	F

Se cumple que $y\#z \leftrightarrow z\#y$ es una tautología, por lo que son lógicamente equivalentes, por lo que el operador # es conmutativo.

Ejercicio 8.

¿Es cierto que en el Cálculo de Enunciados pueden escribirse dos fbs que tengan diferentes letras de proposición y aun así ambas fbs sean lógicamente equivalentes?. En caso de responder positivamente, dar un ejemplo y justificar. En caso de responder negativamente, justificar.

Ayuda: Ver sec. 1.3, Hamilton.

Conclusión de sec. 1.3, Hamilton: En lógica proposicional, lo que importa para la equivalencia lógica es la forma del valor de verdad, no los nombres de las proposiciones. Dos fórmulas son lógicamente equivalentes si tienen la misma tabla de verdad, es decir, coinciden en valor de verdad para todas las combinaciones posibles de sus variables (independientemente de cómo se llamen).

Ejemplo: $(p \rightarrow q)$ es lógicamente equivalente a $(r \rightarrow s)$, ya que podemos decir que $A=(p \rightarrow q)$ y que $B=(r \rightarrow s)$, A y B representan el condicional, por lo que son lógicamente equivalentes.

Ejercicio 9.

Determinar cuáles de las siguientes fbf son lógicamente implicadas por la fbf $(A \wedge B)$.

Fundamentar.

I. A

A es lógicamente implicada por $(A \wedge B)$, ya que si vale $(A \wedge B)$ entonces vale A.

II. $A \leftrightarrow B$

$A \leftrightarrow B$ es lógicamente implicada por $(A \wedge B)$, ya que se cumple que $(A \wedge B) \rightarrow (A \leftrightarrow B)$ es una tautología porque nunca se puede dar el caso en que valga $(A \wedge B)$ pero no $(A \leftrightarrow B)$.

III. $\neg B \rightarrow \neg A$

$\neg B \rightarrow \neg A$ es lógicamente implicada por $(A \wedge B)$, ya que se cumple que $(A \wedge B) \rightarrow (\neg B \rightarrow \neg A)$ es una tautología porque nunca se puede dar el caso en que valga $(A \wedge B)$ pero no $(\neg B \rightarrow \neg A)$.

IV. $A \rightarrow B$

$A \rightarrow B$ es lógicamente implicada por $(A \wedge B)$, ya que se cumple que $(A \wedge B) \rightarrow (A \rightarrow B)$ es una tautología porque nunca se puede dar el caso en que valga $(A \wedge B)$ pero no $(A \rightarrow B)$.

Ayuda: Ver def. 1.7, Hamilton.

Ejercicio 10.

Dada la siguiente tabla de verdad, encontrar tres fbf para cada una que las tenga por tablas de verdad:

I. una fbf (sin restricciones de conectivos).

$$\neg(p \leftrightarrow q)$$

II. una fbf en FNC (forma normal conjuntiva).

Para la FNC se debe ver las filas donde el valor es F:

- $p=V$ y $q=V$.

- $p=F$ y $q=F$.

Entonces, la FNC sería: $(\neg p \vee \neg q) \wedge (p \vee q)$

III. una fbf en FND (forma normal disyuntiva).

Para la FND se debe ver las filas donde el valor es V:

- $p=V$ y $q=F$.

- $p=F$ y $q=V$.

Entonces, la FND sería: $(p \wedge \neg q) \vee (\neg p \wedge q)$

p	q	?
V	V	F
V	F	V
F	V	V
F	F	F

Ayuda: Ver sec. 1.4, Hamilton.

Ejercicio 11.

Explicar porque el siguiente conjunto no es un conjunto adecuado de conectivas $\{\wedge, \vee\}$.

Ayuda: utilizar def. en sec. 1.5, Hamilton.

$\{\wedge, \vee\}$ no es un conjunto adecuado de conectivas ya que al no estar la negación (\neg) no hay manera de simular dicha expresión, por lo que es imposible representar cualquier función de verdad.

Ejercicio 12.

¿Es cierto que si una fbf A es satisfactible entonces A es una tautología”.

Ayuda: utilizar la técnica de demostración por contraejemplo.

No, es falso.

Demostración por contra-ejemplo:

A: $(p \vee q) \rightarrow$ es satisfactible cuando p y/o q son verdaderas. Pero no es una tautología porque cuando ambas son falsas $(p \vee q)$ es falso.

Ejercicio 13.

Sea A una fbf donde aparecen solo los conectivos \wedge, \vee, \neg . Sea A^* la fbf que se obtiene a partir de A reemplazando cada \wedge por \vee y cada \vee por \wedge . Si A es una tautología, A^* ¿también lo es? Justificar.

Ayuda: utilizar la técnica de demostración por contraejemplo.

Esta propiedad no vale, se demuestra por contra-ejemplo:

$$A = (p \vee (\neg p))$$

$$A' = (p \wedge (\neg p))$$

A es tautología, pero A' no lo es.

Ejercicio 14.

Demostrar utilizando la técnica del absurdo que dadas A y B fbf's cualesquiera, siempre ocurre que si A y A \rightarrow B son tautologías entonces B también lo es.

Ayuda: ver prop. 1.9, Hamilton.

Partimos suponiendo que:

A es una tautología.

$(A \rightarrow B)$ es una tautología.

B no es una tautología.

Entonces existe una asignación de valores de verdad a las variables de enunciado que aparecen en A o en B, que da a B el valor F. Pero esta asignación debe dar a el valor V a A ya que A es una tautología, y por tanto da a $(A \rightarrow B)$ el valor F. Esto contradice la hipótesis de que $(A \rightarrow B)$ es una tautología. Por lo tanto, B debe ser una tautología.

Ejercicio 15.

Demostrar utilizando la técnica del absurdo que $(p \wedge \neg p) \rightarrow q$ es una tautología.

Partimos suponiendo que:

$(p \wedge \neg p) \rightarrow q$ no es una tautología, es decir que existe una asignación de valores de verdad que hace que el valor de la forma enunciativa sea F.

El único caso en que $(p \wedge \neg p) \rightarrow q$ es F es cuando q es F y $(p \wedge \neg p)$ es V, por lo que debería haber una asignación de valores de verdad que haga $(p \wedge \neg p)=V$, lo cual es *absurdo*.

Ejercicio 16.

Sea A una fbf donde aparecen solo los conectivos \wedge , \neg . Sea A^* la fbf que se obtiene a partir de A reemplazando cada \wedge por \vee y cada letra de proposición por su negación (o sea, cada p por $\neg p$, cada q por $\neg q$, etc.). Probar, utilizando la técnica de inducción que A^* es lógicamente equivalente a $\neg A$.

Ayuda: ver prop 1.15, Hamilton.

La demostración procede por inducción sobre el número n de conectivas que aparecen en A. Si logramos demostrar que, para cada número natural n, toda forma enunciativa restringida que tenga exactamente n conectivas satisface la proposición, está claro que habremos demostrado todo lo necesario.

Paso base: n=0 (A no contiene conectivas). En este caso A consiste simplemente en una variable de enunciado, p, por ejemplo. Así A^* , en este caso ($\neg p$), de modo que, trivialmente, A^* es lógicamente equivalente a ($\neg A$).

Paso la de inducción: Supongamos que $n > 0$, que A tiene n conectivas, y que toda forma enunciativa con menos de n conectivas posee la propiedad requerida. Debido a las tres maneras de construir formas enunciativas, hemos de considerar tres casos:

Caso 1: A es de la forma ($\neg B$)

Caso 2: A es de la forma ($B \vee C$)

Caso 3: A es de la forma ($B \wedge C$)

Ad Caso 1: B tiene $n-1$ conectivas, así que por hipótesis de inducción B^* es lógicamente equivalente a ($\neg B$). Pero A^* es ($\neg B^*$), así que A^* es lógicamente equivalente a ($\neg(\neg B)$), por ejemplo, a ($\neg A$).

Ad Caso 2: B y C contienen cada una menos de n conectivas, así que B^* y C^* son lógicamente equivalentes a ($\neg B$) y ($\neg C$), respectivamente. Ahora bien, A^* es ($B^* \wedge C^*$). Esto es lógicamente equivalente a ($(\neg B)^* \wedge (\neg C)^*$), y de nuevo esto es lógicamente equivalente a ($(\neg(\neg B)) \wedge (\neg(\neg C))$). Ahora bien, esto es equivalente a ($\neg(B \vee C)$) es decir, ($\neg A$). Así pues, A^* es lógicamente equivalente a ($\neg A$).

Ad Caso 3: Como en el Caso 2, B^* y C^* son lógicamente equivalentes a ($\neg B$) y ($\neg C$), respectivamente. A^* es ($B^* \wedge C^*$), que es lógicamente equivalente a ($(\neg B)^* \wedge (\neg C)^*$) y por tanto a ($(\neg(\neg B)) \wedge (\neg(\neg C))$) y por tanto a ($\neg(B \vee C)$), es decir, a ($\neg A$). Así pues, A^* es lógicamente equivalente a ($\neg A$).

Bajo la hipótesis de que toda forma enunciativa restringida con menos de n conectivas posee la propiedad requerida, hemos demostrado que toda forma enunciativa restringida con n conectivas posee la propiedad requerida. Así pues, por el principio de inducción matemática toda forma enunciativa restringida tiene la propiedad requerida.

Ejercicio 17.

Demostrar utilizando la técnica de inducción que cualquier fórmula bien formada A que contenga sólo los conectivos {V, ∧} puede tomar el valor F.

Ayuda: ver ejercicio resuelto en clase teórica.

Para demostrar que A «puede» tomar el valor falso vamos a construir una valuación en particular en la cual A sea Falsa.

Se hace una valuación, que asigna el valor Falso a todas las letras. O sea $v(p_i)=F$ para todo i. En esa valuación la Fbf A también va a tomar el valor falso.

Para demostrar por inducción necesitamos un número natural N. En este caso el “N” es la “cantidad de conectivos de la Fbf”

CASO BASE. N=0: Como N es cero, no hay conectivos, entonces A es atómica. O sea $A=p_1$ $v(p_1)=F$ por lo tanto $v(A)=F$.

HIPÓTESIS INDUCTIVA (H.I.): asumimos que para toda Fbf A que contiene sólo conjunción y disyunción, con N o menos conectivos $v(A)=F$.

CASO N+1 Usando la H.I. tenemos que poder probar que $v(A)=F$, para una fórmula A que tiene N+1 conectivos. A puede tener 2 formatos:

- Caso 1: A es (B V C).
- Caso 2: A es (B ∧ C).

Vemos que tanto B como C tienen N o menos conectivos, por lo tanto para ellas vale la H.I., es decir $v(B)=F$ y $v(C)=F$. Entonces, por la definición de la semántica del V y del ∧, tanto en el caso 1 como en el caso 2, $v(A)=F$.

Con esto hemos demostrado que $v(A)=F$, para cualquier fbf (que contenga sólo V y ∧).

PRACTICA 9 - Sistema Formal L

Ejercicio 1.

Dada la siguiente demostración sintáctica válida en L:

1. $((\neg p) \rightarrow (\neg(q \rightarrow r))) \rightarrow ((q \rightarrow r) \rightarrow p)$
2. $((\neg p) \rightarrow (\neg(q \rightarrow r)))$
3. $((q \rightarrow r) \rightarrow p)$

a) Identificar el conjunto Γ con menor cantidad de fórmulas bien formadas (fbfs) y la fórmula A tal que $\Gamma \vdash_L A$. Indicar, si es posible, que axioma, hipótesis o regla de inferencia fue aplicado en cada paso de la demostración.

El conjunto $\Gamma = \{(\neg p) \rightarrow (\neg(q \rightarrow r))\}$

La fórmula A = $((q \rightarrow r) \rightarrow p)$

1 es el axioma L3, 2 es hipótesis y 3 se obtiene por la aplicación de MP entre 2 y 1.

b) ¿Es A un teorema de L? Justificar.

No, porque para demostrar A se necesitan la hipótesis, y para que sea un teorema de L necesitaría poder ser demostrada a partir del conjunto vacío, es decir $\vdash_L A$.

c) ¿Es A tautología? Justificar.

No, no es una tautología y se demuestra por contraejemplo:

$$q = V$$

$$r = V$$

$$p = F$$

$\Rightarrow ((q \rightarrow r) \rightarrow p) = F$. Como hemos encontrado al menos un ejemplo en el que $A = F$ podemos asegurar que A no es tautología.

Ejercicio 2.

Sean A , B y C tres fórmulas bien formadas (fbfs) del sistema formal L . Dar una demostración sintáctica en L de las siguientes deducciones. Justificar cada paso en la derivación, indicando cuales son los axiomas instanciados y las reglas de inferencia utilizadas.

Ayuda: es posible utilizar, si es necesario, propiedades ya demostradas en el libro de Hamilton, como por ejemplo, metateorema de la Dedución, silogismo hipotético (SH), y otros teoremas ya demostrados en el libro (ver prop 2.11a y prop 2.11b).

- I. $\vdash_L (((\neg A) \rightarrow A) \rightarrow A)$
 1. $(\neg A \rightarrow A)$ Hipótesis
 2. $(\neg A \rightarrow (\neg(\neg(\neg A \rightarrow A)) \rightarrow \neg A))$ Axioma L_1
 3. $(\neg(\neg(\neg A \rightarrow A)) \rightarrow (\neg A)) \rightarrow (A \rightarrow \neg(\neg A \rightarrow A))$ Axioma L_3
 4. $(\neg A \rightarrow (A \rightarrow \neg(\neg A \rightarrow A)))$ SH entre 2 y 3
 5. $(\neg A \rightarrow (A \rightarrow \neg(\neg A \rightarrow A))) \rightarrow ((\neg A \rightarrow A) \rightarrow (\neg A \rightarrow \neg(\neg A \rightarrow A)))$ Axioma L_2
 6. $(\neg A \rightarrow A) \rightarrow (\neg A \rightarrow \neg(\neg A \rightarrow A))$ MP entre 4 y 5
 7. $(\neg A \rightarrow \neg(\neg A \rightarrow A))$ MP entre 1 y 6
 8. $(\neg A \rightarrow \neg(\neg A \rightarrow A)) \rightarrow ((\neg A \rightarrow A) \rightarrow A)$ Axioma L_3
 9. $(\neg A \rightarrow A) \rightarrow A$ MP entre 7 y 8

- II. $\vdash_L (((\neg B) \rightarrow (\neg A)) \rightarrow (((\neg B) \rightarrow A) \rightarrow B))$
 1. $(\neg B) \rightarrow (\neg A)$ Hipótesis
 2. $((\neg B) \rightarrow (\neg A)) \rightarrow (A \rightarrow B)$ Axioma L_3
 - 3.

- III. $\{(A \rightarrow B) \rightarrow C, B\} \vdash_L (A \rightarrow C)$
 - 1.

Ejercicio 3.

Sea Γ un conjunto de fbfs del sistema formal L . Se sabe que $\Gamma \vdash_L A$ ¿Es cierto que para todo Γ_i tal que $\Gamma_i \subset \Gamma$; $\Gamma_i \vdash_L A$? Fundar.

No es cierto. Se demuestra por contraejemplo:

Si decimos que $\Gamma = \{(A \rightarrow B), C\} \vdash_L A$, entonces $\Gamma = \{(A \rightarrow B)\} \vdash_L A$ no podría ser porque falta C . Entonces así demostramos que se tomamos un $\Gamma_i \subset \Gamma$ y tomamos otro con un elemento menos ya no se cumple por lo que tampoco se cumple todo $\Gamma_i \subset \Gamma$.

Ejercicio 4.

Sea A una fbf y Γ un conjunto de fbfs. Si se cumple $\Gamma \vdash_L A$, ¿Es cierto que vale $\vdash_L A$ para todo A y para todo Γ ? Justificar.

No es cierto. Porque si A se deduce a partir de un conjunto de fbfs, no puede deducirse a partir del conjunto vacío. Se demuestra por contraejemplo.

Si decimos que $\vdash_L A$ con $\Gamma = \{(A \rightarrow B), (\neg C \rightarrow A), B\} \vdash_L A$, entonces no vale $\vdash_L A$ porque faltan $(A \rightarrow B)$, $(\neg C \rightarrow A)$, B ; por lo tanto no se cumple.

Ejercicio 5.

Determinar si las siguientes afirmaciones son válidas o no en el sistema formal L. Justificar en cada caso.

I. $\{q\} \vdash_L (p \rightarrow q)$

Si al **suponer** p podemos deducir q, entonces $\vdash_L (p \rightarrow q)$.

Entonces:

1. q Hipótesis
2. p Hipótesis auxiliar supuesta
3. q Por 1
4. $p \rightarrow q$ Por metateorema de la deducción

II. $\{p \rightarrow q\} \vdash_L (q)$

No es válida. No sabemos el valor de p. Se demuestra por contraejemplo:

$p = F, q = F \Rightarrow (p \rightarrow q) = V$ por lo que se satisface la hipótesis, pero $q = F$, por lo que no se satisface la conclusión.

Ejercicio 6.

Sean A, B y C fbf's del C. de Enunciados. Sea Γ un conjunto de fbf's del C. de Enunciados.

Se sabe que $\Gamma \cup \{A, B\} \vdash_L C$ y también se sabe que $\Gamma \vdash_L A$.

I. ¿Es cierto que $\Gamma \vdash_L (C \rightarrow B)$? Justificar.

II. ¿Es cierto que $\vdash_L (A)$? Justificar.

Tomamos lo siguiente:

$$A = p$$

$$B = p$$

$$\Gamma = p$$

$$C = p$$

$\Gamma \cup \{A, B\} \vdash_L C =$ cuando $v(p) = V \Rightarrow V$, y cuando $v(p) = F \Rightarrow V$

$\Gamma \vdash_L A =$ cuando $v(p) = V \Rightarrow V$, y cuando $v(p) = F \Rightarrow V$.

Pero,

$\vdash_L (A) =$ cuando $v(p) = V \Rightarrow V$, y cuando $v(p) = F \Rightarrow F$,

por lo que se demuestra que $\vdash_L (A)$ no es cierto por contraejemplo.

Ejercicio 7.

¿Es el sistema formal L decidable? Justificar.

Ayuda: si es decidable, debería ser posible determinar (decidir) para cada fbf, si es o no teorema de L.

No, el sistema formal L no es decidable ya que no se puede determinar para cada fbf si es o no teorema de L, porque esto requeriría comprobar que cada fbf sea un teorema de L en tiempo finito, y las fbf's del sistema podrían ser infinitas y llevarnos tiempo exponencial el solo hecho de leer todas y cada una de ellas.

PRACTICA 10 - LOGICA PROPOSICIONAL

Ejercicio 1.

Expresar en un lenguaje de predicados de primer orden las siguientes afirmaciones:

I. Algunas aves no vuelan.

A: es ave.

V: vuela.

$\sim V$: no vuela.

Predicado $\rightarrow (\exists x) (A(x) \rightarrow (\sim V(x)))$

II. No todas las aves vuelan.

A: es ave.

V: vuela.

Predicado $\rightarrow (\sim (\forall x)) (A(x) \rightarrow V(x))$

Analizar la relación entre ambas. Mostrar cómo se puede transformar una expresión en la otra.

A veces decir que existe al menos uno es lo mismo que decir que no todos cumplen con cierta condición.

$\sim (\forall x) \equiv (\exists x)$.

Ejercicio 2.

Expresar en un lenguaje de predicados de primer orden el conocimiento asociado a las siguientes situaciones:

I. Los usuarios que contribuyen en proyectos open source son colaborativos.

C = contribuye en proyecto open source.

K = es colaborativo.

$C(\text{usuarios})$ = Los usuarios contribuyen en proyectos open source.

$K(\text{usuarios})$ = Los usuarios son colaborativos.

u = usuario.

$(\forall u) (C(u) \rightarrow K(u))$

II. Ningún sistema que tenga bugs críticos puede ser entregado ni desplegado en producción.

B = tiene bugs críticos.

E = puede ser entregado.

D = puede ser desplegado en producción.

s = sistema.

$B(s)$ = El sistema tiene bugs críticos.

$E(s)$ = El sistema puede ser entregado.

$D(s)$ = El sistema puede ser desplegado en producción.

$\sim (\exists s) (B(s) \rightarrow (E(s) \wedge D(s)))$

III. Ningún modelo de IA que se entrena con datos erróneos es preciso.

E = se entrena con datos erróneos.

P = es preciso.

m = modelo de IA.

$E(m)$ = Un modelo de IA se entrena con datos erróneos.

$P(m)$ = Un modelo de IA es preciso.

$\sim(\exists m) (E(m) \wedge P(m))$

IV. Todo estudiante que cursa FTC (Fundamentos de Teoría de la computación) y sube sus ejercicios a IDEAS aprueba la práctica.

C = cursa FTC.

S = sube sus ejercicios a IDEAS.

A = aprueba la práctica.

e = estudiante.

$C(e)$ = El estudiante cursa FTC.

$S(e)$ = El estudiante sube sus ejercicios a IDEAS.

$A(e)$ = El estudiante aprueba la práctica.

$(\forall e) ((C(e) \wedge S(e)) \rightarrow A(e))$

V. Todos los alumnos de FTC, cuyo documento es par y han aprobado el parcial con nota mayor a 7 están inscriptos en la mesa de finales de agosto.

C = cursa FTC.

D = el documento es par.

A = aprobó el parcial.

N = la nota es mayor a 7.

I = inscripto en la mesa de finales de agosto.

a = alumno.

$C(a)$ = El estudiante cursa FTC.

$D(a)$ = el documento del alumno es par.

$A(a)$ = el alumno aprobó el parcial.

$N(a)$ = la nota del alumno es mayor a 7.

$I(a)$ = el alumno está inscripto en la mesa de finales de agosto.

$(\forall a) ((C(a) \wedge (D(a) \wedge (A(a) \wedge N(a)))) \rightarrow I(a))$

VI. Todos los estudiantes que cursan FTC y subieron correctamente el código al repositorio están habilitados para correr las pruebas automáticas del sistema.

C = cursa FTC.

S = subio correctamente el código al repositorio.

H = está habilitado para correr las pruebas automáticas del sistema.

e = estudiante.

$C(e)$ = El estudiante cursa FTC.

$S(e)$ = El estudiante subió correctamente el código al repositorio.

$H(e)$ = El estudiante está habilitado para correr las pruebas automáticas del sistema.

$(\forall e) ((C(e) \wedge S(e)) \rightarrow H(e))$

VII. Algunos modelos de inteligencia artificial entrenados por alumnos de FTC lograron superar el umbral de precisión del 90%.

E = entrenado por alumnos de FTC.

L = logró superar el umbral de precisión del 90%.

m = modelo de IA.

$E(m)$ El modelo de IA fue entrenado por alumnos de FTC.

$L(m) =$ El modelo de IA superó el umbral de precisión del 90%.
 $(\exists m) (E(m) \wedge L(m))$

Ejercicio 3.

Escribir las siguientes proposiciones usando un lenguaje de predicados de primer orden:

I. El cero es el menor natural.

$N =$ es natural.

$M =$ es el menor.

$c =$ el cero.

$(N(c) \wedge M(c))$

II. El conjunto vacío está incluido en cualquier conjunto.

$c =$ conjunto.

$cv =$ conjunto vacío.

$I(x, y) = x$ incluye y .

$(\forall c) (I(c, cv))$

III. Si se prueba una propiedad para el cero y luego se prueba que esa misma propiedad vale para el número $n+1$ si vale para n , entonces se ha probado que la propiedad vale para cualquier natural.

$A(x) =$ se prueba x para el cero.

$B(x) =$ se prueba x para el n .

$C(x) =$ se prueba x para $n+1$.

$D(x) =$ se prueba x para cualquier natural.

$P(x) =$ “ x es una propiedad.

Para todo x ($P(x) \wedge (A(x) \wedge (B(x) \rightarrow C(x))) \rightarrow D(x)$)

IV. Si hay un número natural que cumple una cierta propiedad, entonces hay un mínimo natural que cumple esa propiedad.

$C(x) =$ un número natural cumple x .

$M(x) =$ hay un mínimo natural que cumple x .

$p =$ propiedad.

$(C(p) \rightarrow M(p))$

PRACTICA 11 - LÓGICA DE PREDICADOS | SEMÁNTICA

Ejercicio 1.

Señalar las ocurrencias libres o ligadas de x_1, x_2, x_3 en la siguiente fbf escrita en un lenguaje de primer orden donde $C = \{c\}$, $F = \{f, g\}$, y $P = \{A\}$, con f de aridad 1; g de aridad 2, A de aridad 2.

Determinar cual es una fbf abierta y cual es cerrada.

I. $(\forall x_1) ((\exists x_2) A(x_1, f(x_2, x_3)) \rightarrow (\forall x_3) A(g(c), x_1) \vee A(x_1, x_3))$

II. $(\forall x_1) ((\exists x_2) A(x_1, f(x_2, x_3))) \rightarrow (\forall x_3) A(g(c), x_1) \vee A(x_1, x_3)$

Ejercicio 2.

Sea A una fbf que no contiene cuantificadores (es decir, abierta) escrita en algún lenguaje de primer orden. Sea I una interpretación para tal lenguaje. ¿Es posible decidir acerca del valor de verdad de A en I ? Fundamentar.

Si, es posible, ya que el valor de verdad dependerá pura y exclusivamente de la valoración de A en la interpretación I , ya que al ser abierta no tiene cuantificadores, por lo que depende solo de la interpretación.

Cada símbolo y término de la fórmula tiene un significado preciso en I , y por lo tanto, es posible evaluar el valor de verdad de A respecto de I y una asignación σ de variables.

Ejercicio 3.

Analizar si son o no lógicamente equivalentes los siguientes pares de fbfs (usar noción de i-equivalencia o contraejemplos según corresponda):

$$I. (\forall x)P(x) \quad (\exists x)P(x)$$

No son lógicamente equivalentes. Se demuestra por contraejemplo:

Tomando una interpretación I donde el dominio $D = \{1, 2\}$, con:

$$P(1) = V$$

$$P(2) = F$$

Entonces:

$$(\exists x)P(x) \text{ porque existe } P(1) = V.$$

Pero no se cumple $(\forall x)P(x)$ porque existe $P(2) = F$.

Por lo que no son i-equivalentes ya que no se cumple que son equivalentes en todas sus posibles asignaciones.

$$II. (\exists x)(\exists y)Q(x, y) \quad (\exists y)(\exists x)Q(x, y)$$

Si, son lógicamente equivalentes ya que ambos expresan que existe un par de elementos x e y que satisfacen $Q(x, y)$.

Son i-equivalentes ya que para toda asignación de x e y son equivalentes.

$$III. (\exists x)(\forall y)R(x, y) \quad (\forall y)(\exists x)R(x, y)$$

No son lógicamente equivalentes:

En el primer caso dice que “existe un x tal que para todo y se cumple $R(x, y)$ ”,

En el segundo caso dice que “Para cada y existe un x (que puede variar entre las y) que cumple $R(x, y)$ ”.

Se demuestra por contraejemplo:

No son i-equivalentes, definimos un dominio $D = \{1, 2\}$:

x	y	R(x, y)
1	1	V
1	2	F
2	1	F
2	2	V

Para $(\exists x)(\forall y) R(x, y)$:

No hay ningún x tal que $R(x,y)$ sea verdadero para todos los y . Por lo que la fórmula es falsa.

Para $(\forall y)(\exists x) R(x,y)$:

Para $y=1$: $R(1,1)=V$

Para $y=2$: $R(2,2)=V$

$$IV. (\exists x)(S(x) \wedge S(x)) \quad (\exists x)S(x) \wedge (\exists x)S(x)$$

Si, son lógicamente equivalentes, tomando una interpretación I donde el dominio es $D = \{1, 2\}$ se demuestra por la i-equivalecia:

x	S(x)
1	V
2	F

Para $(\exists x)(S(x) \wedge S(x))$:

Para $x = 1$: $(S(x) \wedge S(x)) = V$

Para $x = 2$: $(S(x) \wedge S(x)) = F$

Para $(\exists x)S(x) \wedge (\exists x)S(x)$:

Para $x = 1$ en ambos casos: $S(x) = V \Rightarrow (\exists x)S(x) \wedge (\exists x)S(x) = V$

Para $x = 2$ en ambos casos o $x = 1$ en el primero y $x = 2$ en la segunda $\Rightarrow (\exists x)S(x) \wedge (\exists x)S(x) = F$

Entonces, ambas expresiones son lógicamente equivalentes, ya que coinciden en sus valores en todas las ocurrencias de posibles valores, V con $x = 1$ siempre, y F en el resto.

$$V. (\exists x)(T(x) \vee T(x)) \quad (\exists x)T(x) \vee (\exists x)T(x)$$

No, no son lógicamente equivalentes, y se demuestra por contrajeemplo, tomando una interpretación I donde el dominio es $D = \{1, 2\}$:

x	T(x)
1	V
2	F

Para $(\exists x)(T(x) \vee T(x))$:

Para $x = 1$: $T(x) \vee T(x) = V$

Para $x = 2$: $T(x) \vee T(x) = F$

Para $(\exists x)T(x) \vee (\exists x)T(x)$:

Para $x = 2$ en ambos casos $\Rightarrow (\exists x)T(x) \vee (\exists x)T(x) = F$

Para $x = 1$ en ambos casos, o $x = 1$ en uno y $x = 2$ en otro $\Rightarrow (\exists x)T(x) \vee (\exists x)T(x) = V$

Por lo que no son i-equivalentes, ya que no tienen los mismos valores en todas las ocurrencias.

$$VI. (\forall x)(U(x) \vee U(x)) \quad (\forall x)U(x) \vee (\forall x)U(x)$$

Si, son lógicamente equivalentes, ya que son i-equivalentes. Tomando una interpretación I donde el dominio es $D = \{1, 2\}$:

x	U(x)

1	V
2	F

Para $(\forall x)(U(x) \vee U(x))$:

Para $x = 1$: $U(x) \vee U(x) = V$

Para $x = 2$: $U(x) \vee U(x) = F$

Para $(\forall x)U(x) \vee (\forall x)U(x)$:

Para $x = 1$ en ambas, $U(x) \vee U(x) = V$

Para $x = 1$ en una y $x = 2$ en otra, $U(x) \vee U(x) = V$

Para $x = 2$ en ambas, $U(x) \vee U(x) = F$

Por lo que son i-equivalentes, ya que tienen siempre los mismos valores en todas las posibles valoraciones.

Ejercicio 4.

Sea un lenguaje de primer orden con las siguientes características:

- **Conjunto de constantes:** $C = \{c, u\}$
- **Sin símbolos de función:** $F = \emptyset$
- **Conjunto de símbolos de predicado:** $P = \{A\}$.

Sea I la siguiente interpretación para ese lenguaje sobre el dominio de los números

Naturales:

- $I(c) = 0$
- $I(u) = 1$
- $I(A(x, y)) = "x \leq y"$

Verificar si las siguientes afirmaciones son o no correctas. Justificar las respuestas.

- $A(c, x)$ es satisfactible en I .

Como x es c , entonces si $I(c) = 0 \rightarrow I(x) = 0$,
entonces $A(0, 0) = "0 \leq 0"$ es satisfactible ya que $0 = 0$.

- $A(u, x)$ es satisfactible en I .

Como x es u , entonces si $I(u) = 1 \rightarrow I(x) = 1$,
entonces $A(1, 1) = "1 \leq 1"$ es satisfactible ya que $1 = 1$.

- $(\forall x) A(c, x)$ es satisfactible en I .

Si, es satisfactible en I porque $I(x)$ es siempre $I(c) = 0$, por lo que x siempre igual a 0.

- $(\forall x) A(u, x)$ es satisfactible en I .

Si, es satisfactible en I porque $I(x)$ es siempre $I(c) = 1$, por lo que x siempre igual a 1.

- $A(c, x)$ es verdadera en I .

Si, es verdadera. Ya que $I(x)$ toma el valor de $I(c)$ y c es constante, por lo que se satisface en todos los casos y que todos los casos es $c = 0$.

- $(\forall x)A(c, x)$ es lógicamente válida.

No, porque depende de cual sea la interpretación de c, x, y $A(x, y)$ en las demás interpretaciones. Eso es solo válido en nuestra interpretación I .

- $A(u, c) \wedge \neg A(u, c)$ es contradictoria.

Si, ya que no importa cual sea la interpretación de $A(x, y)$, u y c, la conjunción de $A(u, c)$ y su negación nunca será verdadera.

Ejercicio 5.

Ofrecer una interpretación donde las siguientes fórmulas sean todas verdaderas y otra donde sean falsas. Traducir en cada caso las fórmulas dadas a oraciones apropiadas en lenguaje natural.

Todas verdaderas:

- $x = 0$.
- $y = 1$.
- $z = 2$.
- $c = 3$.
- $P(a, b) = "a \leq b"$.
- $f(x) = x + x$.

Todas falsas:

- Constantes a, b.
- $c = a$.
- $P(x, y) = "x \neq y"$.
- $f(x) = b$ si $x = a$, a si $x = b$.

I. $(\forall x) P(x, x)$

V: “Para todo x, x es $\leq x$ ”

II. $\neg((\forall x)(\forall y)(P(x, y) \rightarrow P(y, x)))$

V: “No vale que para todo x y para todo y, si x es $\leq y$ entonces y es $\leq x$ ”

III. $(\forall x)(\forall y)(\forall z)((P(x, y) \wedge P(y, z)) \rightarrow P(x, z))$

V: “Para todo x, para todo y y para todo z, vale que si x es $\leq y$ e y es $\leq z$, entonces x es $\leq z$ ”

IV. $(\forall x)P(c, x) \vee (\forall x)P(x, f(x))$

“Para todo x vale que c es $\leq x$ o para todo x vale que x es $\leq x + x$ ”

V. $(\forall x)P(x, f(x))$

“Para todo x vale que x es $\leq x + x$ ”

Ejercicio 6.

Determinar para cada una de las siguientes fórmulas escritas en algún lenguaje de primer orden si son satisfactibles en alguna interpretación, verdaderas en alguna interpretación, falsas en alguna interpretación, lógicamente válidas o contradictorias.

Fundamentar.

I. $(\forall x)P(x)$

Satisfactible: Interpretación I donde x es constante = 0 y $P(x) = "x = 0"$.

Verdadera: Interpretación I donde x es constante = 0 y $P(x) = "x = 0"$.

Falsa: Interpretación I donde x es constante = 0 y $P(x) = "x > 0"$.

II. $((\forall x)(\forall y)Q(x, y)) \rightarrow Q(x, y)$

Es **Lógicamente Válida** ya que en el único caso donde esta fbf sería falsa es en aquel donde “para todo x y para todo y vale siempre Q(x, y)” y Q(x, y) es falsa, cosa que no puede darse nunca, por lo que esta fbf es verdadera en todas las interpretaciones.

III. $(\exists x)(\exists y)Q(x, y) \rightarrow (\exists y)(\exists x)Q(x, y)$

Satisfactible: Interpretación I donde $x = \{0, 1, 2\}$, $y = \{0, 1, 2\}$, $Q(x, y) = "x = y"$.

Verdadera: Interpretación I donde $x = \{0, 1, 2\}$, $y = \{3, 4, 5\}$, $Q(x, y) = "x < y"$.

Falsa: Interpretación I donde $x = \{0, 1, 2\}$, $y = \{3, 4, 5\}$, $Q(x, y) = "x > y"$.

IV. $Q(x) \rightarrow Q(x)$

Es **Lógicamente Válida** ya que si $Q(x)$ es verdadera entonces la fbf es verdadera, y si $Q(x)$ es falsa la fbf es verdadera.

V. $(\exists x)(\neg P(x)) \vee (\forall x)(P(x) \vee Q(x))$

Ejercicio 7.

Determinar si las siguientes fbf's son (o no) lógicamente válidas o contradictorias.

Fundamentar en cada caso.

I. $((\forall x)(P(x) \vee Q(x))) \rightarrow ((\forall x)P(x) \vee (\forall x)Q(x))$

No son ni lógicamente válidas ni contradictorias, ya que son satisfactibles sólo en algunas interpretaciones. Por ejemplo:

Interpretación I:

- Dominio D = {a}
- a = {0, 1, 2, 3}
- $P(x) = "x = 0"$
- $Q(x) = "x > 0"$
- No se cumple $(\forall x)P(x)$.
- No se cumple $(\forall x)Q(x)$.
- Se cumple $(\forall x)(P(x) \vee Q(x))$.

II. $P(x, y) \rightarrow P(x, y)$

Lógicamente Válida, ya que si $P(x, y)$ es verdadera, la fbf es verdadera, y si $P(x, y)$ es falsa la fbf es verdadera.

III. $P(x, y) \rightarrow (\forall x)(\forall y)(P(x, y))$

No es lógicamente válida ni contradictoria, ya que si $P(x, y)$ es verdadera la fbf se satisface, y si $P(x, y)$ es falsa la fbf es no se satisface.

Ejercicio 8.

Si la fbf $P(x)$ es satisfactible, ¿entonces la fbf $(\exists x)P(x)$ es lógicamente válida?

Fundamentar.

No, porque que la fbf $P(x)$ sea satisfactible no implica que en todas y cada una de las interpretaciones la fbf $P(x)$ se satisfaga. Solo implica que existe un x en al menos una interpretación donde la fbf se satisface.

PRÁCTICA 12 - INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

Ejercicio 1.

a) Defina Inteligencia e Inteligencia Artificial.

Inteligencia: Capacidad de adquirir conocimientos, razonar, resolver problemas, planificar, aprender y adaptarse a nuevas situaciones.

Inteligencia Artificial (IA): Rama de la informática que busca desarrollar sistemas capaces de realizar tareas que requieren inteligencia humana, como razonar, aprender, reconocer patrones, comprender lenguaje natural, y tomar decisiones.

b) De ejemplos de sistemas de IA actuales.

ChatGPT, Google Translate, Reconocimiento Facial (Face ID), Sistemas de Recomendación, Asistentes Virtuales (Siri, Alexa).

c) Explique la diferencia entre IA débil e IA fuerte.

IA débil (o estrecha): Diseñada para resolver **una tarea específica** (ej: jugar ajedrez, responder preguntas). No tiene conciencia ni comprensión real.

IA fuerte: Posee capacidad cognitiva general, es decir, puede razonar y aprender como un humano en múltiples dominios. Implica conciencia, autonomía y comprensión real.

d) Explique la diferencia entre un sistema de IA y un sistema de software complejo (como el que calcula la trayectoria de un cohete a Marte).

Un sistema que calcula trayectorias a Marte sigue **leyes físicas exactas** y no necesita aprender, mientras que un sistema de IA que reconoce enfermedades debe aprender de **datos y errores**.

Sistema de IA	Sistema de software complejo
Aprende y mejora con la experiencia	Funciona con reglas fijas y preprogramadas
Puede manejar incertidumbre	Necesita datos precisos
Ejemplo: reconocimiento de imágenes	Ejemplo: simulación física de trayectorias
Toma decisiones en entornos variables	Suele operar en entornos deterministas

e) Defina IA simbólica e IA no-simbólica.

IA simbólica:

- Basada en **lógica formal y reglas explícitas**.
- Representa conocimiento mediante **símbolos y reglas**.
- Ejemplo: sistemas expertos, razonadores lógicos.

IA no-simbólica:

- Usa estructuras **numéricas o estadísticas**.
- Aprende comportamientos sin reglas explícitas.
- Ejemplo: redes neuronales, algoritmos evolutivos.

f) Defina Machine Learning (ML). ¿ML está incluido en IA?

Machine Learning (ML) es una rama de la IA que se basa en el desarrollo de algoritmos que **aprenden automáticamente a partir de datos**, sin ser explícitamente programados para cada tarea.

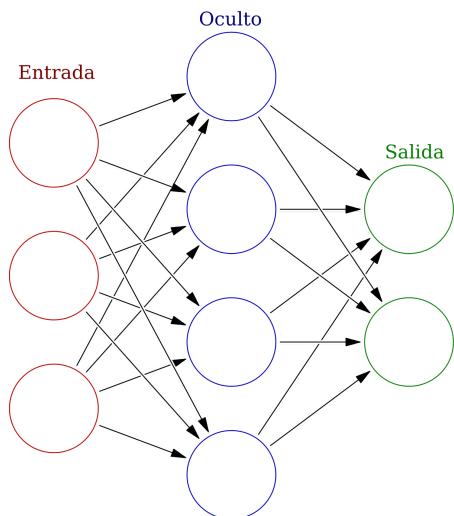
Sí, ML es una **subárea dentro de la IA**, especialmente de la IA **no-simbólica**.

g) ¿Que es una Red neuronal artificial (ANN)? Dibuje un ejemplo señalando cada una de sus partes.

Es un modelo computacional inspirado en el **funcionamiento del cerebro humano**.

Se compone de **neuronas artificiales (nodos)** conectadas entre sí.

Se utiliza para tareas como clasificación, reconocimiento de patrones, predicción, etc.



Ejercicio 2.

Lea el artículo original de Turing sobre IA (Turing 1950). En él se comenta algunas objeciones potenciales a su propuesta y a su prueba de inteligencia. ¿Cuáles de estas objeciones tiene todavía validez? ¿Son válidas sus refutaciones? ¿Se te ocurren nuevas objeciones a esta propuesta teniendo en cuenta los desarrollos realizados desde que se escribió el artículo?

Todavía tienen validez las objeciones de que las máquinas no tienen conciencia ni sentimientos, ya que actualmente la IA solo procesa lenguaje y patrones pero no tiene conciencia ni comprensión real; la objeción de que las máquinas no pueden originar nada y solo hacen lo que se les ordena, ya que todo lo que las IAs actuales generan, lo generan a partir de datos existentes sin intencionalidad, creatividad consciente ni propósito.

Si, la mayoría de sus refutaciones fueron válidas y anticiparon el ML moderno.

Nuevas objeciones que se podrían plantear:

- La IA puede imitar el lenguaje sin entenderlo.
- Las IAs son frágiles al contexto o ambigüedad, ya que muchas veces fallan en tareas de sentido común que los humanos resuelven sin esfuerzo.
- Las IAs tienen una gran dependencia de los datos ya que dependen de grandes volúmenes de datos que les generan sesgos o falsas generalizaciones.

En el artículo, Turing predijo que para el año 2000 sería probable que una computadora tuviera un 30 por ciento de posibilidades de superar una Prueba de Turing dirigida por un

evaluador inexperto con una duración de cinco minutos. ¿Consideras razonable lo anterior en el mundo actual?

Si, es razonable la predicción, ya que para el año 2000 quizás el porcentaje se encontraba por debajo del 30%, pero ya hoy modelos como ChatGPT pueden **engañosamente engañar a evaluadores inexpertos**, aunque no tienen comprensión real.

PRÁCTICA 13 - VERIFICACIÓN AXIOMÁTICA DE PROGRAMAS

Ejercicio 1.

Responder breve y claramente los siguientes incisos:

1.1. ¿En qué se diferencia una prueba semántica de una prueba sintáctica de un programa?

Una prueba semántica utiliza las instrucciones del lenguaje y busca que el mismo sea coherente, mientras que una prueba sintáctica utiliza axiomas y reglas de un método lógico-deductivo.

1.2. ¿Qué es un estado de un programa? ¿Cuándo un estado satisface un predicado?

Un **estado** es una asignación de valores a todas las variables del programa en un momento dado.

Un estado **satisface un predicado** si, al reemplazar las variables en el predicado con los valores del estado, el **predicado resulta verdadero**.

1.3. ¿Cómo se especifica un programa?

Un programa se especifica mediante una **precondición P** y una **postcondición Q**, formando una **triple de Hoare**: {P} S {Q}.

Donde:

P → condición que debe cumplirse antes de ejecutar el programa S.

Q → condición que debe cumplirse después, si S termina.

1.4. ¿Cuál es el significado de la fórmula {p} S {q}?

{p} S {q} significa que si el programa S comienza en un estado que cumple la precondición P, y S **termina**, entonces el estado final **satisfará** la postcondición Q.

1.5. ¿Qué son el invariante y el variante de un while?

El **invariante** es una condición lógica que se cumple antes del bucle, que se mantiene durante todas las iteraciones, y que es crucial para demostrar la corrección parcial.

El **variante** es una expresión que toma valores en un conjunto bien ordenado (nros naturales), disminuye en cada iteración, y sirve para demostrar la terminación (corrección total).

1.6. ¿Cuándo un método axiomático es sensato, y cuándo es completo?

Un método axiomático es sensato si todo lo que puede demostrarse sintácticamente también es semánticamente válido.

Un método axiomático es completo si toda propiedad que es semánticamente válida puede ser demostrada sintácticamente.

La sensatez asegura que las pruebas son válidas; la completitud, que no falta nada demostrable.

1.7. ¿Por qué la lógica de Hoare para los programas secuenciales es composicional?

Porque permite razonar sobre partes del programa de forma independiente, y luego componer esas pruebas parciales para verificar todo el programa.

Si verifico $\{P\} S_1 \{Q\}$ y $\{Q\} S_2 \{R\}$, entonces puedo deducir $\{P\} S_1 : S_2 \{Q\}$.

Ejercicio 2.

Especificar un programa que a partir de un estado inicial en el que x sea mayor que 0, termine en un estado final en el que y sea el cuadrado del valor inicial de x , y además x tenga su valor inicial.

$$\{x = X \wedge x > 0\} \subseteq \{y = X^2, x = X\}$$

$\{x = X \wedge x > 0\} \vee := x * x \{y = X^2, x = X\}$

Ejercicio 3

Decir y justificar informalmente (es decir sin usar la lógica de Hoare), si se cumplen o no las siguientes fórmulas

Comentario: el predicado true representa cualquier estado:

3.1. $\{x > 0\}$ while $x > 0$ do $x := x - 2$ od $\{\text{true}\}$

Si se cumple, ya que al tener ‘true’ como postcondición alcanza con llegar a un estado final cualquiera, es decir llegar a que el while finalice, y eso sucede con cualquier $x > 0$.

3.2. $\{x > 0\}$ while $x > 0$ do $x := x - 2$ od $\{x = 0\}$

No se cumple la siguiente fórmula. Contraejemplo: $x = 1$, al hacer $x := x - 2$, $x = -1$. Termina el while pero no se cumple la postcondición.

3.3. $\{x > 0\}$ while $x \neq 0$ do $x := x - 2$ od $\{\text{true}\}$

No se cumple. Contraejemplo: $x = 1$, al hacer $x := x - 2$, $x = -1$. El while pasa de largo por el 0, x seguirá disminuyendo infinitamente y nunca se alcanzará ni siquiera un estado final.

Ejercicio 4.

Probar (usando la lógica de Hoare): $\{ \text{true} \} x := 0 ; x := x + 1 ; x := x + 2 \{ x = 3 \}$.

De derecha a izquierda:

- | | | |
|---|------|---|
| 1. $\{x = 1\} x := x + 2 \{x = 3\}$ | ASI | \rightarrow Si $x + 2 = 3$, entonces, antes de ejecutar eso, $x = 1$. |
| 2. $\{x = 0\} x := x + 1 \{x = 1\}$ | ASI | \rightarrow Si $x + 1 = 1$, entonces, antes de ejecutar eso, $x = 0$. |
| 3. $\{\text{true}\} x := 0 \{x = 0\}$ | ASI. | |
| 4. $\{\text{true}\} x := 0 ; x := x + 1 \{x = 1\}$ | SEC | \rightarrow De 3 y 2. |
| 5. $\{\text{true}\} x := 0 ; x := x + 1 ; x := x + 2 \{x = 3\}$ | SEC | \rightarrow De 4 y 1. |

Ejercicio 5.

Se cumple $\{x = 10\}$ while $x > 0$ do $x := x - 1$ od $\{x = 0\}$. Se pide probar (usando la lógica de Hoare) que el predicado $p = (x \geq 0)$ es un invariante del while.

Ayuda: hay que probar, por un lado: $x = 10 \rightarrow p$, y por otro lado: $\{p \wedge x > 0\} x := x - 1 \{p\}$.
 $x = 10 \rightarrow x \geq 0$:

- $$1. \quad \{x = X \wedge X = 10\} \ x := X \{x \geq 0\} \quad \text{ASI.}$$

$\{x \geq 0 \wedge x > 0\} x := x - 1 \{x \geq 0\}$:

1.

Ejercicio 6.

La instrucción *repeat S until B* consiste en ejecutar S, luego evaluar B, si B es falsa volver a iterar, y en caso contrario terminar. Explicar informalmente por qué la siguiente regla para probar la instrucción es sensata (es decir, si se cumplen las premisas, entonces también se cumple la conclusión):

$$\frac{\{p\} S \{q\}, q \wedge \neg B \rightarrow p}{\{p\} \text{repeat } S \text{ until } B \{q \wedge B\}}$$

La siguiente regla es sensata porque a partir de las premisas verdaderas se cumple la conclusión. En ambas se ejecuta S a partir de $\{p\}$, se evalúa si se cumplió B, y si no se cumplió continuamos en $\{p\}$ hasta que B se cumpla. Por esto, a partir de premisas verdaderas obtenemos conclusiones verdaderas y la conversión mantiene el significado de la expresión.

La regla es **sensata** porque:

- Garantiza que cada iteración parte de una condición válida p,
- Que el cuerpo mantiene la propiedad deseada q,
- Y que, mientras no se cumple B, el estado sigue siendo válido para continuar,
- Finalmente, cuando B se cumple, se garantiza que el estado cumple $q \wedge B$, como requiere la postcondición.