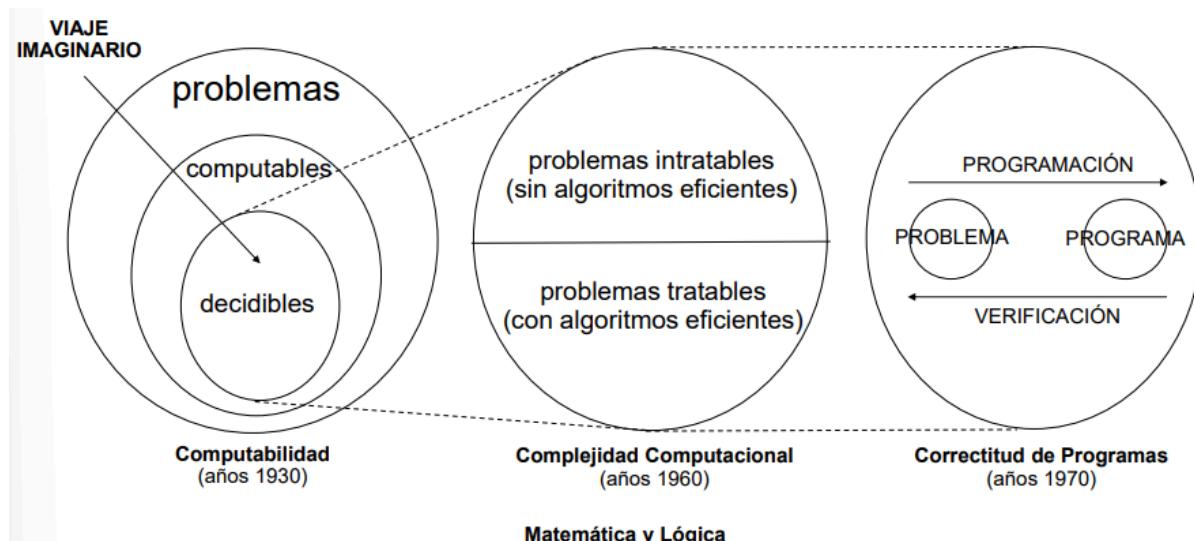


RESUMEN FTC

Clase 1 | La Maquina de Turing

INTRODUCCIÓN



. De la totalidad de los problemas tenemos un subgrupo que pueden ser computables, y dentro de ellos solo algunos pueden ser decidibles.

Esta fue la primer definición de computabilidad dada en 1930.

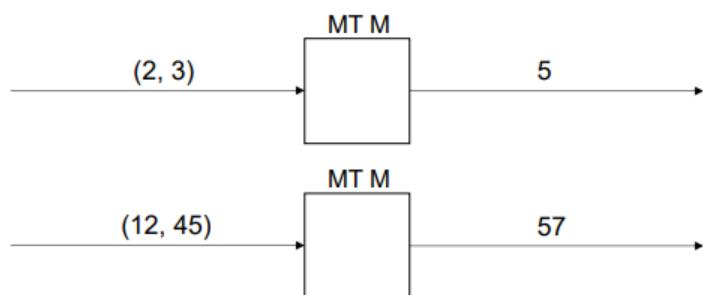
Dentro de los problemas decidibles, la mitad son problemas intratables y la otra mitad tratables. Esta clasificación data de los años 60', donde aparece la Complejidad Computacional.

Por último, para los años 70' nace el concepto de Correctitud de Programas, donde se separa el problema del programa, y donde la programación va del problema al programa y la verificación del programa al problema.

MÁQUINA DE TURING

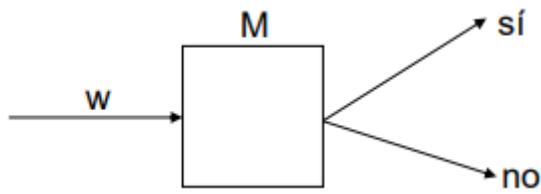
Modelización muy simple de una computadora, aceptada universalmente para estudiar la computabilidad y la complejidad computacional de los problemas.

Ejemplo. MT M que resuelve el problema de la suma de dos números naturales (enteros ≥ 0):

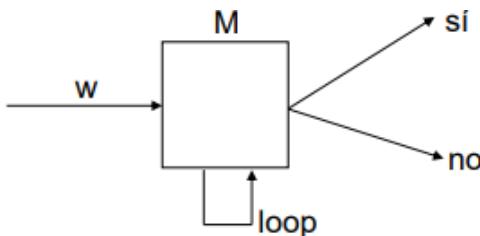


M se comporta de esta manera a partir de todo par de números naturales.

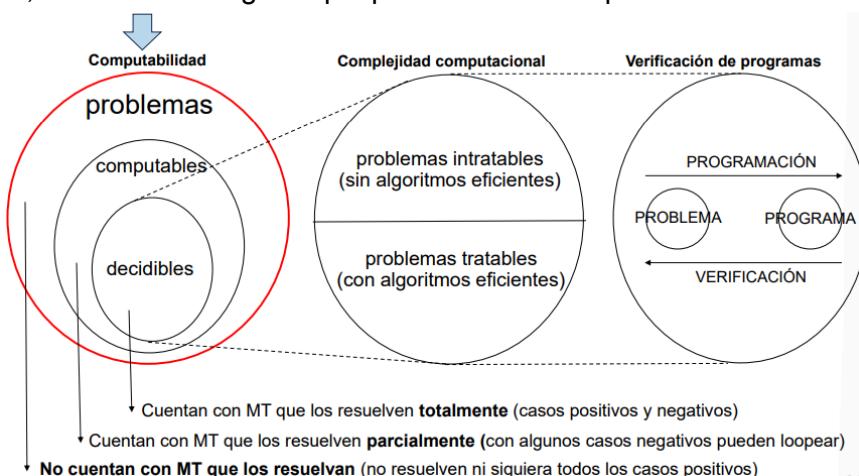
Una Maquina de Turing, inicialmente, si entiende el lenguaje y puede resolverlo devolverá el resultado, si no entiende el lenguaje responderá con "no".



En un caso más general, una MT corresponde a un esquema donde en el mejor de los casos si la entrada no pertenece al lenguaje responderá “no”, pero en el peor de los casos “loopea”.



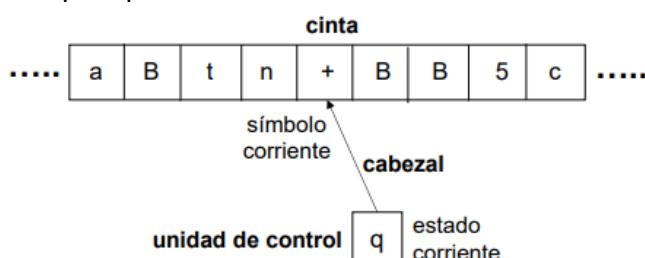
En otros casos, no existe MT alguna que pueda resolver el problema.



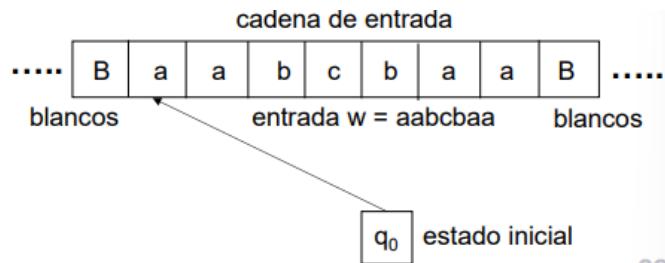
. Componentes:

Una MT consta de una cinta donde se almacenan los símbolos de un alfabeto *gama*, una unidad de control que tiene siempre un estado de un conjunto de estados *Q*.

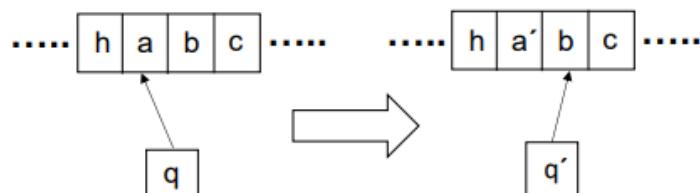
Además tiene un cabezal que apunta a una celda de la cinta.



Al inicio, la entrada se delimita por blancos, y el cabezal apunta al primer símbolo de la izquierda:



En un paso, una MT puede: modificar el símbolo corriente, modificar el estado corriente, y moverse un lugar a la derecha o a la izquierda. Por ejemplo:

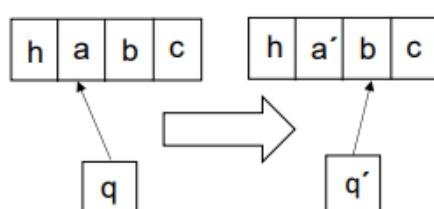


Formalmente, una MT M es una **tupla** $(Q, \Gamma, \delta, q_0, q_A, q_R)$:

- Q es el conjunto de estados de M.
- Γ es el alfabeto (conjunto de símbolos) de M. Γ incluye al símbolo blanco B. Las cadenas de entrada no admiten blancos.
- q_0 es el estado inicial de M.
- q_A y q_R son los estados finales de M (de aceptación y de rechazo).
- δ es la función de transición de M (especifica su comportamiento): Dado un estado corriente de Q, y un símbolo corriente de Γ , la MT M:
 - pasa eventualmente a un nuevo estado de Q,
 - modifica eventualmente el símbolo corriente de Γ ,
 - y se mueve un lugar a la derecha (R), a la izquierda (L), o no se mueve (S).

La máquina para si en algún momento alcanza el estado q_A o q_R .

$$\delta(q, a) = (q', a', R)$$



A partir de toda máquina que acepta un lenguaje hay una que lo genera.

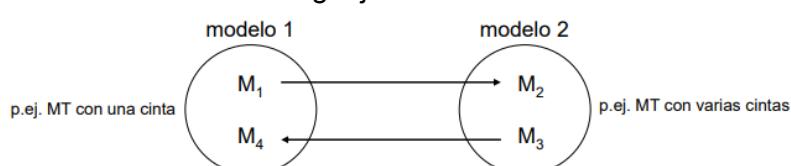
. *Tesis de Church-Turing*:

Todo dispositivo computacional físicamente realizable puede ser simulado por una MT.

. *MT equivalentes*:

Dos máquinas son equivalentes si reconocen (o deciden) el mismo lenguaje).

Dos modelos de MT son equivalentes si todo lenguaje que puede reconocer (o decidir) un modelo, también lo puede el otro. Tienen el mismo poder computacional, es decir, reconocen exactamente los mismos lenguajes.



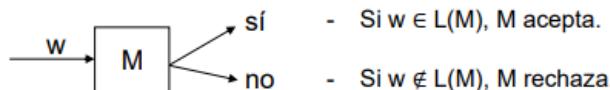
. Visiones:

Visión Calculadora: calcula la posible solución de un problema.

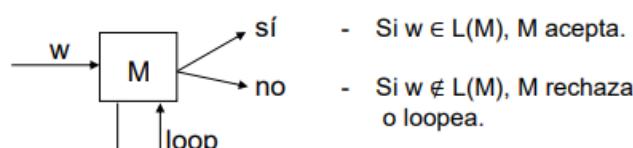
Visión Generadora: genera posibles cadenas que pertenezcan al lenguaje dado.

Clase 2 | La jerarquía de la computabilidad

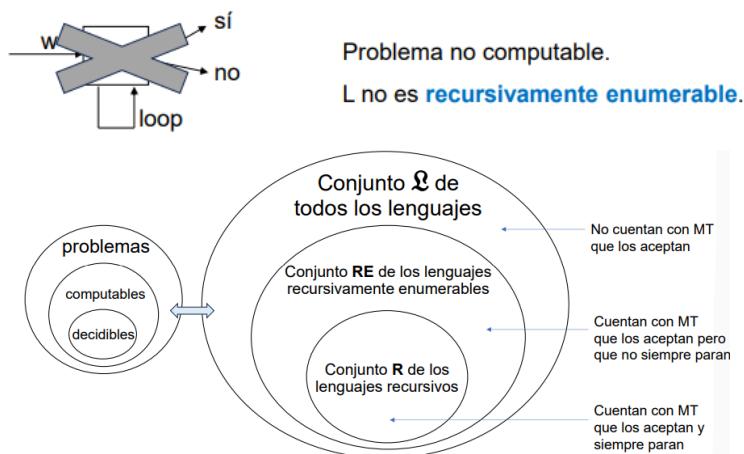
. Problemas **computables decidibles**: el L es *recursivo*. Trabaja con lenguajes aceptados por MT que siempre paran. R es el conjunto de estos lenguajes.



. Problemas **computables no decidibles**: el L es *recursivamente enumerable*. Trabaja con lenguajes aceptados por M que a partir de algunas instancias negativas no paran. RE es el conjunto de estos lenguajes.



. Problemas **no computables**: el L no es *recursivamente enumerable*. Son lenguajes sin MT que los acepten, es decir que a partir de algunas instancias positivas no paran.



ALFABETOS, CONJUNTOS Y LENGUAJES

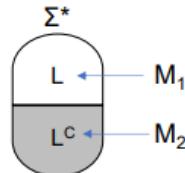
- Σ es el alfabeto universal de todos los símbolos: $\Sigma = \{a, b, \dots, 1, 2, \dots, +, -, \dots\}$
- Σ^* es el conjunto universal de todas las cadenas de símbolos de Σ : $\Sigma^* = \{\lambda, a, b, 1, \dots, aa, ab, a1, \dots, aaa, \dots\}$
- L es el conjunto universal de todos los lenguajes con alfabeto Σ : conjunto de todos los subconjuntos de Σ^* .
- Un lenguaje L es recursivo ($L \in R$) si existe una MT M que lo acepta y siempre para (lo decide). Es decir, para toda cadena w de Σ^* :
 - Si $w \in L$, entonces M a partir de w para en su estado qA
 - Si $w \notin L$, entonces M a partir de w para en su estado qR
- Un lenguaje L es recursivamente enumerable ($L \in RE$) si existe una MT M que lo acepta. Es decir, para toda cadena w de Σ^* :
 - Si $w \in L$, entonces M a partir de w para en su estado qA

- Si $w \notin L$, entonces M a partir de w para en su estado qR o no para.
- Se cumple por definición que $R \subseteq RE \subseteq \Omega$. Las inclusiones son estrictas: $R \subset RE \subset \Omega$.

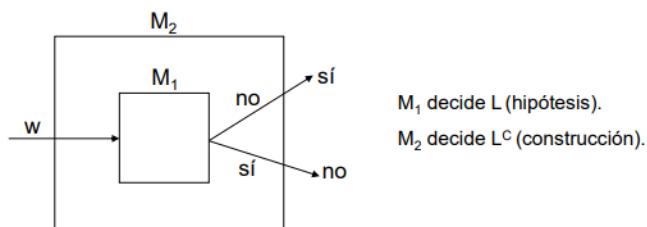
LA CLASE R

Propiedad 1: Si $L \in R$, entonces $L(\text{complemento}) \in R$.

Es decir, si existe una MT M_1 que decide L , también existe una MT M_2 que decide $L(\text{complemento})$.



Idea general: Construir una MT M_2 que responda al revés que la MT M_1 .



Construcción de la MT m_2 :

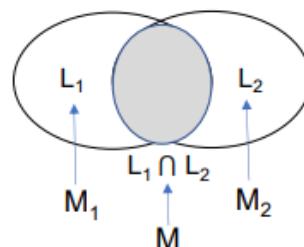
$$\text{Si: } M_1 = (Q, \Gamma, \delta, q_0, q_A, q_R)$$

$$\text{entonces: } M_2 = (Q, \Gamma, \delta', q_0, q_A, q_R)$$

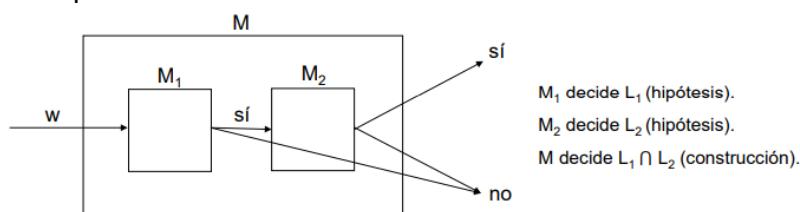
donde las funciones de transición δ y δ' de M_1 y M_2 son iguales salvo que los estados qA y qR están permutados.

Propiedad 2: Si $L_1 \in R$ y $L_2 \in R$, entonces $L_1 \cap L_2 \in R$.

Es decir, 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 $L_1 \cap L_2$.



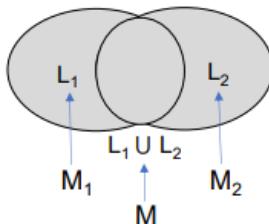
Idea general: Construir una MT M que ejecute secuencialmente las MT M_1 y M_2 y acepte si y sólo si M_1 y M_2 aceptan.



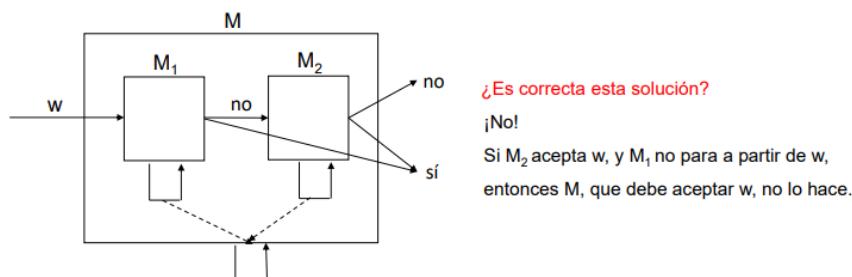
LA CLASE RE

Propiedad 3: Si $L_1 \in RE$ y $L_2 \in RE$, entonces $L_1 \cup L_2 \in RE$.

Es decir, 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 $L_1 \cup L_2$.

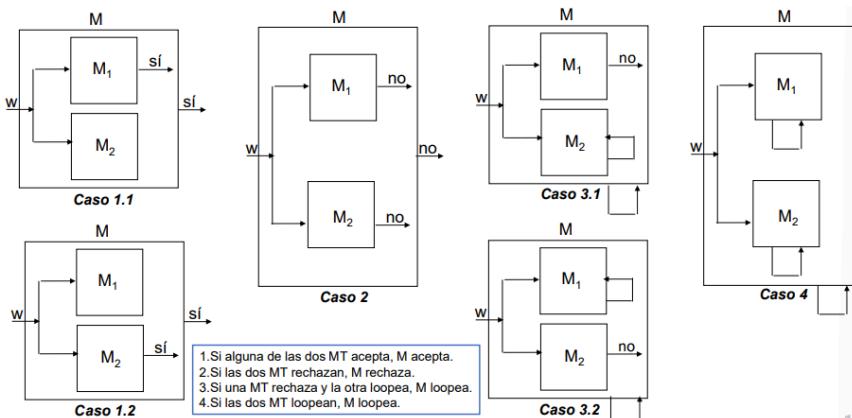


Idea general: ¿Construir una MT M que ejecute secuencialmente las MT M_1 y M_2 y acepte si M1 o M2 aceptan?



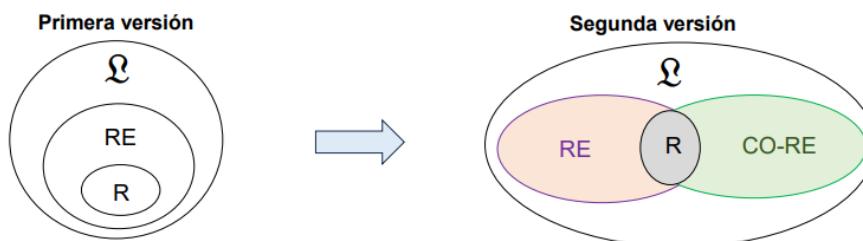
La forma correcta es ejecutar “en paralelo” M_1 y M_2 , y aceptar w si una de las dos MT acepta w .

“En paralelo” M_1 y M_2 = ejecutar un paso de M_1 y un paso de M_2 alternadamente.



12

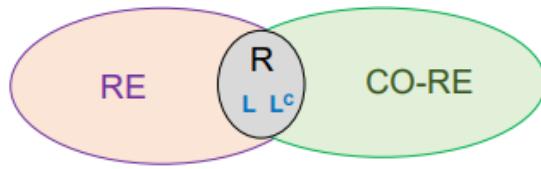
SEGUNDA VERSIÓN DE LA JERARQUÍA DE LA COMPUTABILIDAD



El conjunto CO-RE tiene los complementos de los lenguajes del conjunto RE ($L \in RE$ si $L_{(\text{complemento})} \in CO-RE$).

La utilidad de CO-RE es proveer información acerca de que su complemento es computable. Es su única diferencia con el conjunto que se encuentra por fuera de R, RE y CO-RE.

- Claramente se cumple $R \subseteq RE \cap CO-RE$:



$R \subseteq RE$ por definición.

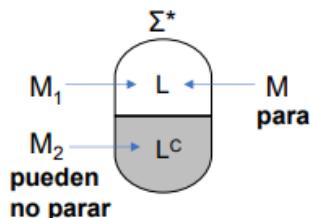
$R \subseteq CO-RE$ porque: si $L \in R$, entonces también $L_{(complemento)} \in R$, por lo que $L_{(complemento)} \in RE$, y así por definición: $L \in CO-RE$.

Propiedad 4: $RE \cap CO-RE = R$.

Sólo falta probar la inclusión $RE \cap CO-RE \subseteq R$.

- Hay que probar:

- si existe una MT M_1 que acepta L ,
- y existe una MT M_2 que acepta $L_{(complemento)}$,
- también existe una MT M que acepta L y siempre para (decide L).

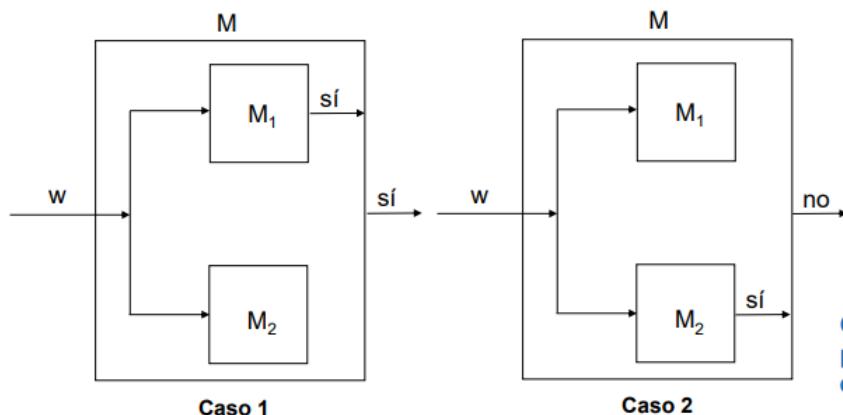


Idea general: M ejecuta en paralelo M_1 y M_2 .

Si M_1 acepta w , M acepta w .

Si M_2 acepta w , M rechaza w .

Cualquiera sea w , M_1 la acepta o M_2 la acepta. ¿Por qué? Porque toda cadena w pertenece a L o a $L_{(complemento)}$.



Conclusión: si un problema y el problema contrario son computables, entonces ambos son decidibles.

Las cuatro regiones de la jerarquía de la computabilidad

Región 1 (lenguajes con MT que siempre paran).

Conjunto R.

Si L está en R, entonces $L_{(complemento)}$ está en R.

Región 2 (lenguajes con MT).

Conjunto RE – R.

Si L está en RE, entonces $L_{(\text{complemento})}$ está en CO-RE.

Región 3 (lenguajes sin MT, con complementos con MT).

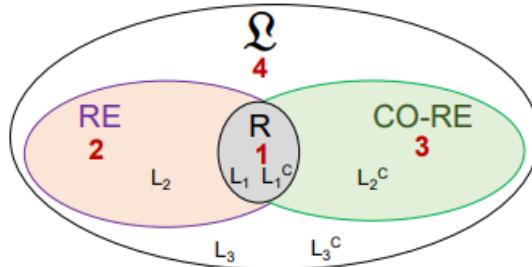
Conjunto CO-RE – R.

Si L está en CO-RE, entonces $L_{(\text{complemento})}$ está en RE.

Región 4 (lenguajes sin MT, con complementos sin MT).

Conjunto $\mathfrak{L} – (\text{RE} \cup \text{CO-RE})$.

Si L está en $\mathfrak{L} – (\text{RE} \cup \text{CO-RE})$, entonces $L_{(\text{complemento})}$ está en $\mathfrak{L} – (\text{RE} \cup \text{CO-RE})$.



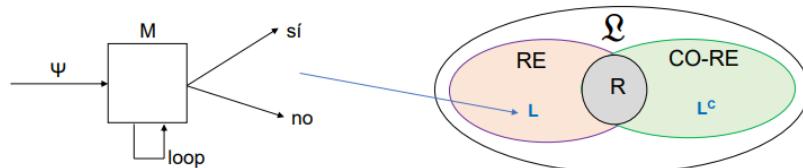
Las cuatro regiones de la jerarquía,
con grado de dificultad creciente.

PROBLEMA DE LA RESOLUCIÓN DE LAS ECUACIONES DIOFÁNTICAS

Dada una ecuación algebraica con coeficientes enteros y variables enteras (ecuación diofántica), como por ejemplo $2x^3 + 5y^3 = 6z^3$, ¿la ecuación tiene solución? El lenguaje que representa el problema es:

$$L = \{\Psi \mid \Psi \text{ es una ecuación diofántica y tiene solución}\}$$

Se prueba que $L \in \text{RE} - \text{R}$.



Ejercicio: ¿cómo funcionaría M?

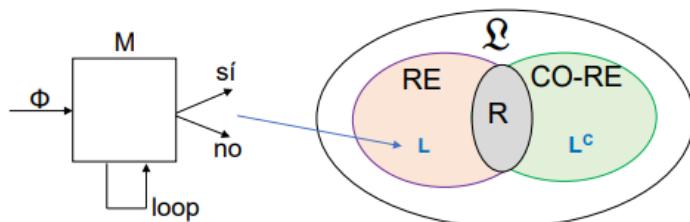
Con p.ej. ecuaciones diofánticas de la forma $x^n + y^n = z^n$ con $n \geq 0$, el problema es decidible.

PROBLEMA DE DECISIÓN EN LA LÓGICA DE PREDICADOS (LP)

¿Existe una prueba de la fórmula Φ en la LP? (Φ es un teorema?). El lenguaje que representa el problema es:

$$L = \{\Phi \mid \text{existe una prueba de la fórmula } \Phi \text{ en la LP}\}$$

Se prueba que $L \in \text{RE} - \text{R}$.



Ejercicio: ¿cómo funcionaría M?

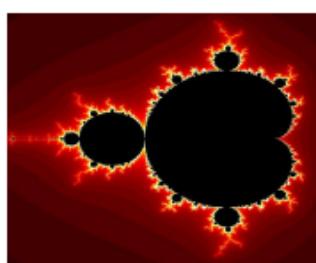
Con fórmulas de cierto tipo, el problema es decidible.

PROBLEMA DE PERTENENCIA AL CONJUNTO DE MANDELBROT (CM)

El CM es un conjunto de números complejos del plano definido por la sucesión $z_0 = 0$ y $z_{n+1} = z_n^2 + c$, tal que c está en el conjunto si la sucesión está acotada. El problema es: dado un número complejo c , ¿ c está en el CM?

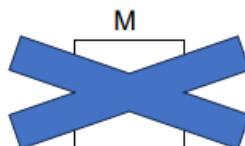
El lenguaje asociado es $L = \{c \mid c \text{ es un número complejo que está en el CM}\}$.

Se prueba que $L \in \text{CO-RE} - R$.



Conjunto de Mandelbrot (es un fractal)

Hay elementos del CM que una MT no puede reconocer: La dificultad está en el contorno del conjunto.



Pero al menos existe una MT M que acepta todos los números complejos que NO están en el CM:

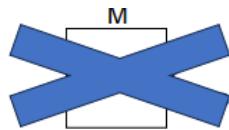


PROBLEMA DE LA DECISIÓN EN LA ARITMÉTICA

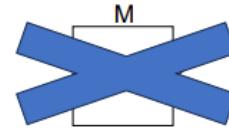
¿La fórmula Θ es un enunciado aritmético verdadero? El lenguaje que representa el problema es:

$$L = \{\Theta \mid \Theta \text{ es un enunciado aritmético verdadero}\}.$$

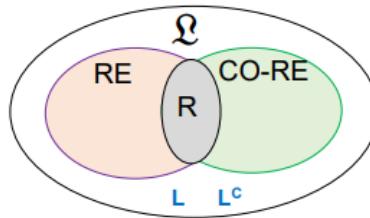
Se prueba que $L \in \mathfrak{L} - (\text{RE} \cup \text{CO-RE})$.



Hay enunciados verdaderos que una MT no puede reconocer.



Hay enunciados falsos que una MT no puede reconocer.



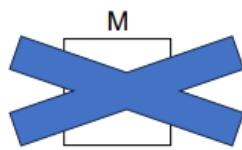
Sin la multiplicación el problema es decidible.

PROBLEMA DEL CUBRIMIENTO DEL PLANO CON POLÍGONOS (TESELACIÓN DEL PLANO)

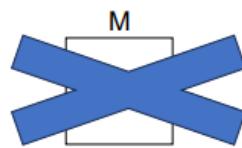
Dado un conjunto finito C de figuras poligonales (conocidas como teselas o mosaicos), ¿C puede cubrir el plano sin dejar huecos ni producir solapamientos? El lenguaje que representa el problema es:

$$L = \{C \mid C \text{ es un conjunto finito de figuras poligonales que cubren el plano}\}$$

Se prueba que $L \in \Omega - (RE \cup CO-RE)$.



Hay conjuntos de polígonos que cubren el plano que una MT no puede reconocer.

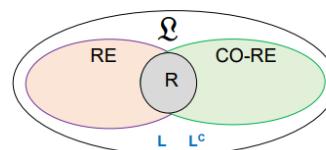


Hay conjuntos de polígonos que no cubren el plano que una MT no puede reconocer.



EJEMPLO DE TESELACIÓN

Nota: si hay periodicidad, el problema es decidible.



22

Clase 3 | Indecibilidad

Debemos probar $R \subset RE \subset \Omega$:

Hay lenguajes en $RE - R$ (región 2): hay problemas computables no decidibles.

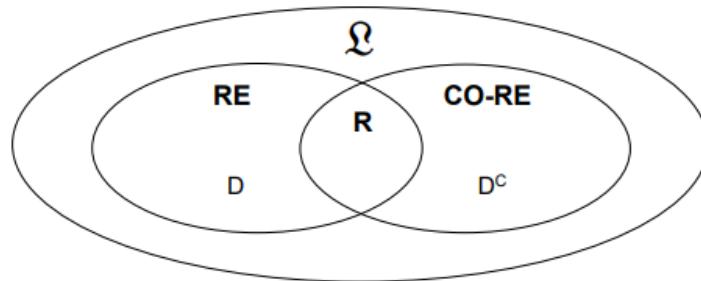
Hay lenguajes en $L - RE$ (regiones 3 y 4): hay problemas no computables.

Utilizaremos el método de diagonalización para probar no pertenencia a R o RE (construyendo MT no se puede probar que un lenguaje NO pertenece a una clase de lenguajes).

Vamos a encontrar:

Un primer lenguaje D en $RE - R$, para probar $R \subset RE$.

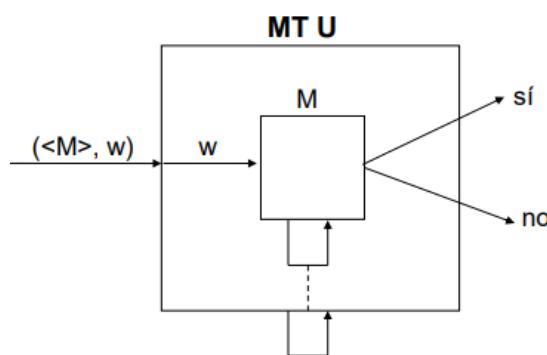
Un primer lenguaje DC en $CO-RE - R$, para probar $RE \subset \Omega$.



Para ello, primero tenemos que introducir la máquina de Turing universal.

. Máquina de Turing Universal:

Una máquina de Turing universal (MT U) es una máquina de Turing capaz de ejecutar cualquier otra MT (noción de programa almacenado, Turing 1936). El esquema más general es:



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

PRUEBA DE QUE $R \subset RE \subset \Omega$

La siguiente tabla T representa el comportamiento de todas las MT M con respecto a todas las cadenas w:

T	todas las cadenas						
	w ₀	w ₁	w ₂	w ₃	w ₄	
fila 0 M ₀	1	0	1	1	1	
fila 1 M ₁	1	0	0	1	0	
fila 2 M ₂	0	0	1	0	1	
fila 3 M ₃	0	1	1	1	1	
fila 4 M ₄	0	1	1	1	0	
.....	

todas las MT

$$T(M_i, w_j) = 1 \text{ si } M_i \text{ acepta } w_j; T(M_i, w_j) = 0 \text{ si } M_i \text{ rechaza } w_j \text{ (los valores son de ejemplo)}$$

La fila 0 = (1, 0, 1, 1, 1, ...), representa el lenguaje $L(M_0) = \{w_0, w_2, w_3, w_4, \dots\}$.

La fila 1 = (1, 0, 0, 1, 0, ...), representa el lenguaje $L(M_1) = \{w_0, w_3, \dots\}$.

La fila 2 = (0, 0, 1, 0, 1, ...), representa el lenguaje $L(M_2) = \{w_2, w_4, \dots\}$.

Etc.

Por lo tanto, las filas representan todos los lenguajes aceptados por MT, es decir el conjunto: RE.

Consideremos en particular la diagonal de la tabla T:

T	w ₀	w ₁	w ₂	w ₃	w ₄
fila 0 M ₀	1	0	1	1	1
fila 1 M ₁	1	0	0	1	0
fila 2 M ₂	0	0	1	0	1
fila 3 M ₃	0	1	1	1	1
fila 4 M ₄	0	1	1	1	0
.....

- La diagonal es (1, 0, 1, 1, 0, ...), y representa el lenguaje D = {wi | Mi acepta wi }. En el ejemplo: D = {w0 , w2 , w3 , ...}.
- Y la diagonal con los 1 y 0 invertidos es (0, 1, 0, 0, 1, ...), y representa el lenguaje DC = {wi | Mi rechaza wi }. En el ejemplo: DC = {w1 , w4 , ...}.

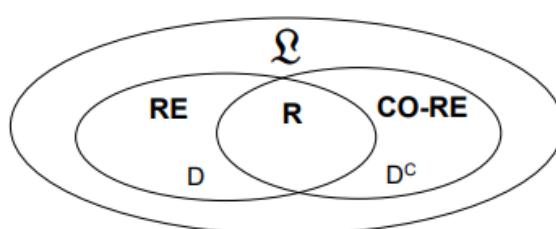
Entonces:

(1) D está en RE

(2) D(complemento) no está en RE

Y por lo tanto: (3) D no está en R ¿Por qué?

Porque si D está en R, D(complemento) está en RE.

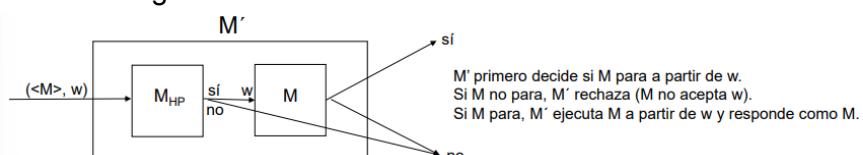


A partir de estos primeros lenguajes se pueden encontrar otros lenguajes fuera de R y RE, con un método más sencillo, la reducción.

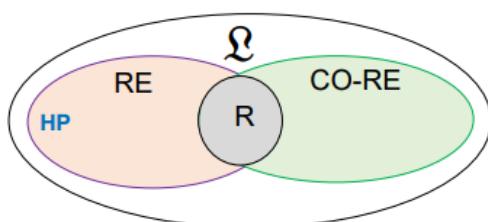
HALTING PROBLEM

HP = {(<M>, w) | M para a partir de w} está entre los lenguajes más difíciles de RE.

Una forma de verlo es probando que si HP fuera recursivo, entonces todo lenguaje de RE sería recursivo. Sea M_{HP} una MT que decide HP y M una MT que reconoce algún lenguaje L de RE. La siguiente MT M' decide L:

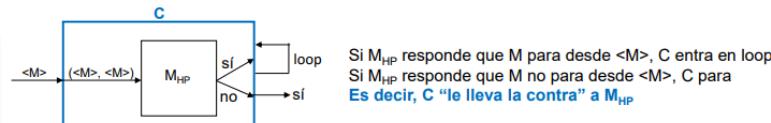


HP está en la frontera de RE, lo más lejos posible de R. Se dice que es RE-completo, identifica el grado de dificultad de RE, representa el problema general de la indecidibilidad.

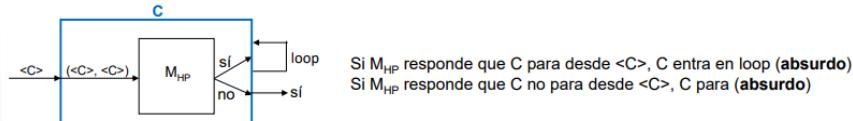


LA PRUEBA DE TURING DE QUE EL LENGUAJE HP NO ES RECURSIVO

Supondremos que $HP = \{<M>, w \mid M \text{ para a partir de } w\} \in R$ y llegaremos a una contradicción. Sea M_{HP} una MT que decide HP. Utilizando M_{HP} podemos construir la siguiente MT C:



Veamos qué sucede cuando la entrada de C es su propio código $<C>$:



Así, C no puede existir, y cómo se construyó a partir de M_{HP} , tampoco M_{HP} puede existir. Por lo tanto, $HP \notin R$.

Clase 4 | Reducciones

Para probar pertenencia a R y RE hemos construido MT.

Para probar no pertenencia a R y RE hemos utilizado el método de diagonalización.

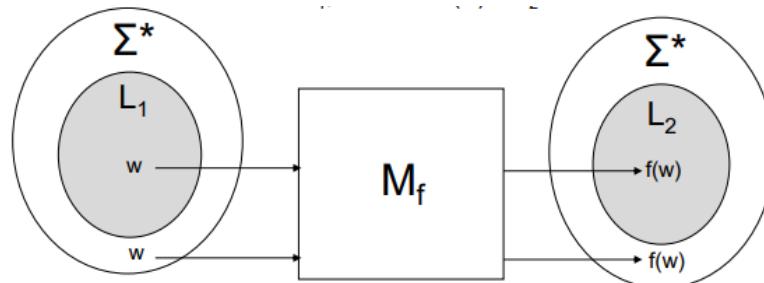
Pero hay otro método más sencillo que la diagonalización para probar no pertenencia a R y RE: *la reducción*.

REDUCCIONES DE LENGUAJES

Reducir es fabricar o construir una MT que lleve un elemento dentro de L_1 a un elemento dentro de L_2 , y un elemento fuera de L_1 a uno fuera de L_2 .

Dados dos lenguajes L_1 y L_2 , una reducción de L_1 a L_2 es una función total computable $f : \Sigma^* \rightarrow \Sigma^*$ (función definida para todas las cadenas de Σ^* y computable por una MT M_f), que para toda cadena w :

$$\begin{aligned} &\text{si } w \in L_1, \text{ entonces } f(w) \in L_2 \\ &\text{si } w \notin L_1, \text{ entonces } f(w) \notin L_2 \end{aligned}$$



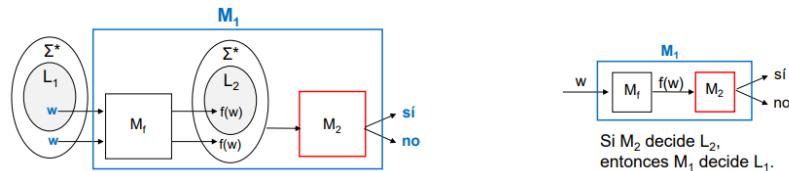
El objetivo es poder reducir (relacionar) el lenguaje L_1 al lenguaje L_2 , con la idea de construir una MT M_1 que acepte L_1 utilizando una MT M_2 ya conocida que acepta L_2 , en lugar de construir M_1 de cero.

Esto se asemeja a la invocación de una subrutina, ya que reutilizamos algo que ya está hecho, anda y está probado.

La notación $L_1 \leq L_2$ expresa que existe una reducción de L_1 a L_2 .

. Caso 1 - $L_2 \in R$:

En lugar de construir de cero una MT M1 para decidir L_1 , se la puede construir a partir de una MT M2 conocida que decide L_2 (noción de invocación a una subrutina en programación).



Formalmente: si $L_1 \leq L_2$, entonces $L_2 \in R \rightarrow L_1 \in R$.

O lo mismo: si $L_1 \leq L_2$, entonces $L_1 \notin R \rightarrow L_2 \notin R$.

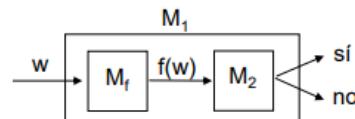
Encontrando una reducción de un lenguaje L_1 que no está en R a un lenguaje L_2 , se prueba que L_2 tampoco está en R .

Caso 1

Si $L_1 \leq L_2$ entonces ($L_2 \in R \rightarrow L_1 \in R$)

O bien, por el contrarrecíproco:

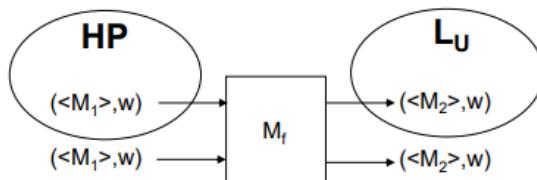
Si $L_1 \leq L_2$ entonces ($L_1 \notin R \rightarrow L_2 \notin R$)



Ejemplo:

$HP = \{(\langle M \rangle, w) \mid M \text{ para a partir de } w\}$ y $LU = \{(\langle M \rangle, w) \mid M \text{ acepta } w\}$.

Vamos a probar $HP \leq LU$. Idea general:



Sabemos que: si $L_1 \leq L_2$, entonces $L_1 \notin R \rightarrow L_2 \notin R$
 Así, de $HP \leq LU$ y $HP \notin R$, probamos $LU \notin R$

Reducción: M2 es como M1, salvo que los estados qR de M1 se cambian en M2 por estados qA.

Computabilidad: Mf copia $(\langle M_1 \rangle, w)$ pero cambiando los estados qR de M1 por estados qA en M2.

Correctitud:

$(\langle M_1 \rangle, w) \in HP \rightarrow M_1 \text{ para desde } w \rightarrow M_2 \text{ acepta } w$, ya que si M1 para qA M2 dejará qA, mientras que si para por qR M2 lo cambia por qA $\rightarrow (\langle M_2 \rangle, w) \in LU$.

$(\langle M_1 \rangle, w) \notin HP \rightarrow$ si $(\langle M_1 \rangle, w)$ es una cadena válida:

M1 no para desde w $\rightarrow M_2$ no para desde w, ya que solo cambia qR por qA, pero no contempla el caso en que loopee $\rightarrow (\langle M_2 \rangle, w) \notin LU$.

si $(\langle M_1 \rangle, w)$ no es una cadena válida:

$(\langle M_2 \rangle, w)$ tampoco es una cadena válida $\rightarrow (\langle M_2 \rangle, w) \notin LU$.

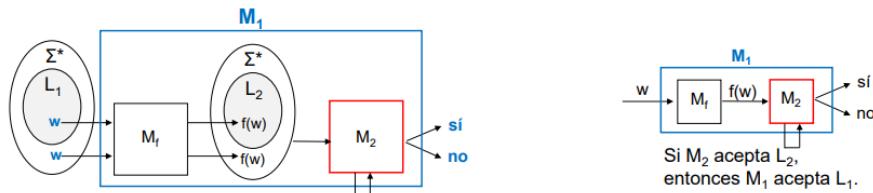
La función de identidad ($f(w)=w$) no es una reducción de HP a LU ya que para que esto se cumpla debe $(\langle M \rangle, w) \in HP$ si y sólo si $f(\langle M \rangle, w) \in LU$.

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.

. Caso 2 - $L \in RE$:

En lugar de construir de cero una MT M1 para aceptar L1, se la puede construir a partir de una MT M2 conocida que acepta L2 (noción de invocación a una subrutina en programación).



Si $w \in L_1$, entonces $f(w) \in L_2$, entonces M_2 acepta $f(w)$, entonces M_1 acepta w .

Si $w \notin L_1$, entonces $f(w) \notin L_2$, entonces M_2 rechaza $f(w)$ (puede loopear), entonces M_1 rechaza w (puede loopear).

Formalmente: si $L_1 \leq L_2$, entonces $L_2 \in RE \rightarrow L_1 \in RE$.

O lo mismo: si $L_1 \leq L_2$, entonces $L_1 \notin RE \rightarrow L_2 \notin RE$.

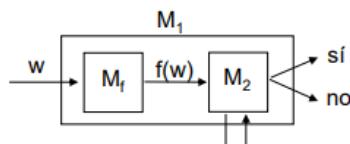
Encontrando una reducción de un lenguaje L_1 que no está en RE a un lenguaje L_2 , se prueba que L_2 tampoco está en RE.

Caso 2

Si $L_1 \leq L_2$ entonces ($L_2 \in RE \rightarrow L_1 \in RE$)

O bien, por el contrarrecíproco:

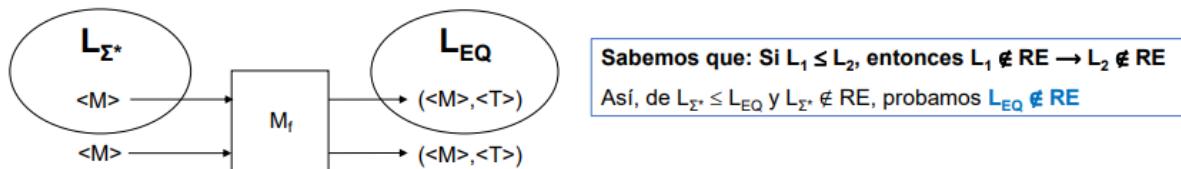
Si $L_1 \leq L_2$ entonces ($L_1 \notin RE \rightarrow L_2 \notin RE$)



Ejemplo:

$L_{\Sigma^*} = \{<M> \mid L(M) = \Sigma^*\}$ y $LEQ = \{(<M_1>, <M_2>) \mid L(M_1) = L(M_2)\}$.

Vamos a probar $L_{\Sigma^*} \leq LEQ$. Idea general (comentario: se prueba que $L_{\Sigma^*} \notin RE$):



Reducción: es una MT que acepta el lenguaje Σ^* : para cada cadena que genera en el orden canónico, corre $<M>$ sobre cada una y si acepta, acepta.

Computabilidad: Mf copia $<M>$ y le concatena $<T>$.

Correctitud:

$<M> \in L_{\Sigma^*} \rightarrow L(M) = \Sigma^* \rightarrow L(M) = L(T) \rightarrow (<M>, <T>) \in LEQ$

$<M> \notin L_{\Sigma^*} \rightarrow$ si es una cadena válida:

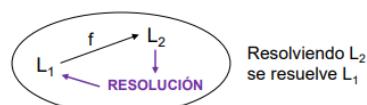
$L(M) \neq \Sigma^* \rightarrow L(M) \neq L(T) \rightarrow (<M>, <T>) \notin LEQ$.

si no es una cadena válida:

$(<M>, <T>)$ tampoco es una cadena válida $\rightarrow (<M>, <T>) \notin LEQ$.

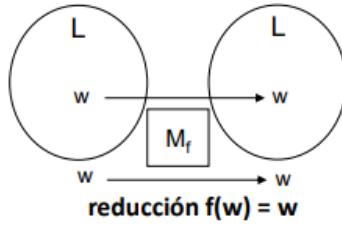
Conclusión:

Si $L_1 \leq L_2$, entonces L_2 es tan o más difícil que L_1
Si $L_1 \notin R$, no puede suceder que $L_2 \in R$
Si $L_1 \notin RE$, no puede suceder que $L_2 \in RE$

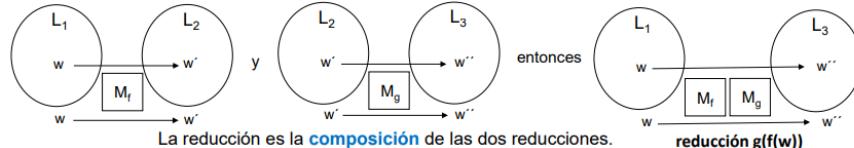


. Propiedades:

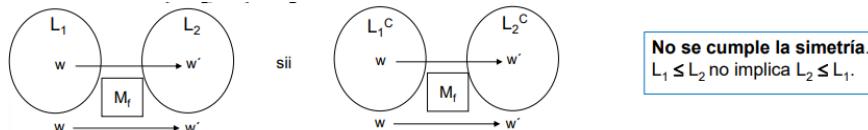
Reflexividad. Para todo lenguaje L se cumple $L \leq L$. La reducción es la función identidad.



Transitividad. Si $L_1 \leq L_2$ y $L_2 \leq L_3$, entonces $L_1 \leq L_3$.

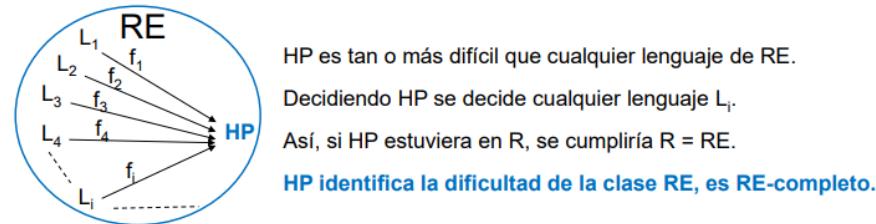


Otra propiedad: $L_1 \leq L_2$ si $L_1^C \leq L_2^C$. La reducción es la misma.



. HP con Reducción:

Se prueba que todo lenguaje L de RE cumple $L \leq \text{HP}$:



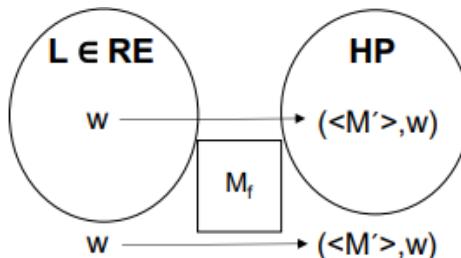
Idea general de la prueba:

Sea una MT que acepta L , M' es como M , salvo que desde todo qR de M se incluye en M' un loop.

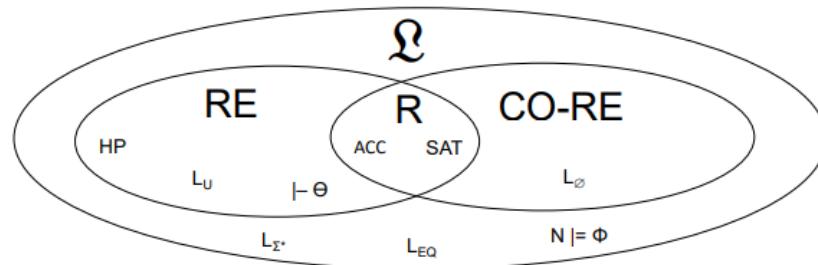
De esta manera:

$w \in L \rightarrow M$ acepta $w \rightarrow M'$ para a partir de w .

$w \notin L \rightarrow M$ rechaza $w \rightarrow M'$ no para a partir de w .



PROBLEMAS CLÁSICOS DE LA COMPUTABILIDAD



ACC, problema de accesibilidad: ¿El grafo no dirigido G tiene un camino de su primer a su último vértice?

SAT, problema de satisfactibilidad: ¿La fórmula booleana φ es satisfactible?

HP, problema de la parada: ¿La MT M para a partir de la cadena de entrada w ?

LU, problema de aceptación universal: ¿La MT M acepta la cadena de entrada w ?

⊤-Θ, problema de decisión en la lógica de predicados (LP): ¿La fórmula Θ es un teorema de la LP?

L_∅: ¿El lenguaje aceptado por la MT M es el lenguaje vacío?

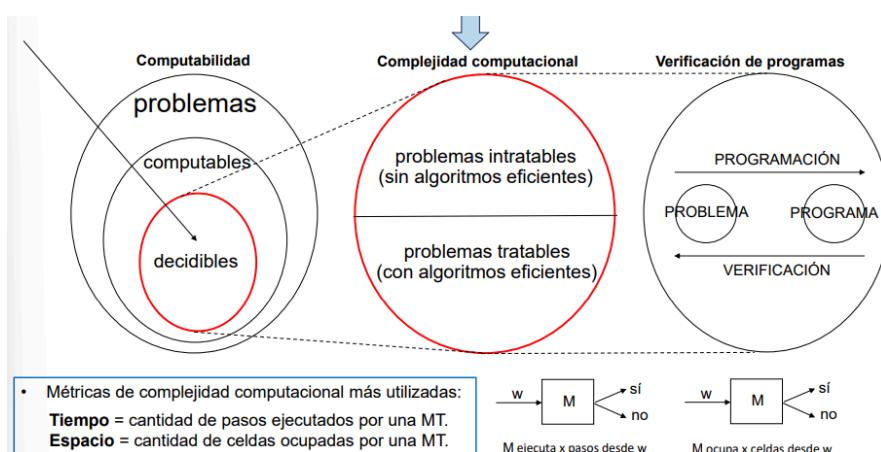
L_{Σ*}: ¿El lenguaje aceptado por la MT M es el lenguaje Σ^* ?

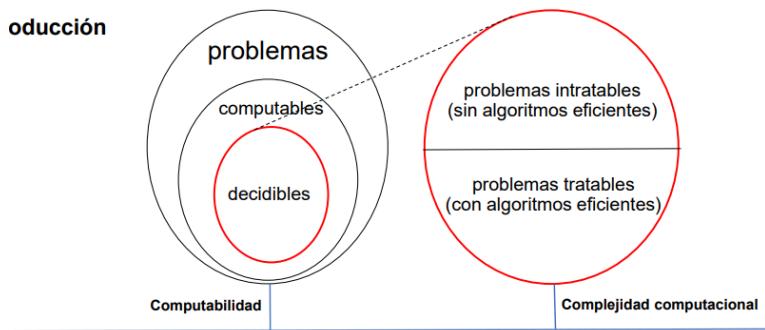
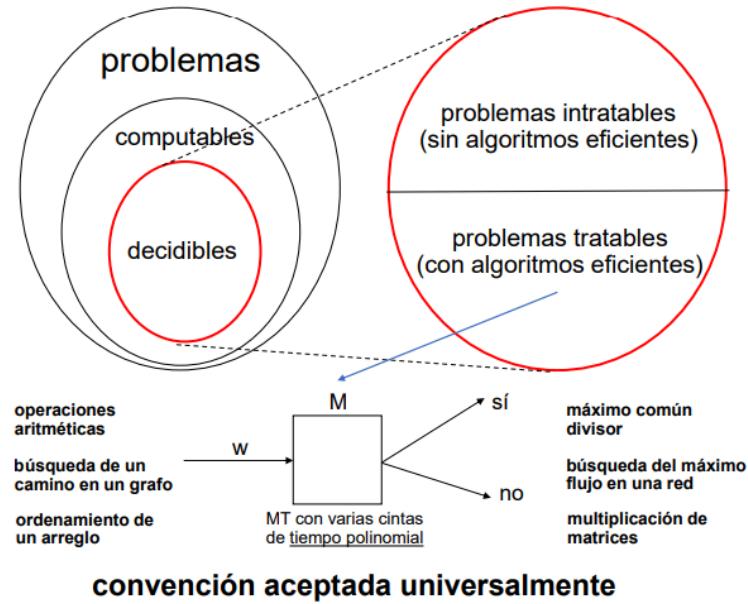
LEQ, problema de equivalencia: ¿Las MT M y M' son equivalentes?

N |= Φ, problema de decisión en la aritmética: ¿El enunciado aritmético Φ es verdadero?

* Ver ANEXO (Clase 4).

Clase 5 | Tiempo Polinomial y No Polinomial





En ambos casos:

- **Análisis estructural:** enfocado no en problemas particulares sino en clases de problemas.
- **Objetivo:** entender por qué algunos problemas son más difíciles que otros.
- **Escenario de estudio:** MT con varias cintas como modelo computacional, y pruebas por medio de la construcción de MT, la diagonalización y las reducciones.

COMPLEJIDAD TEMPORAL

- . Una MT M tarda más (hace más pasos) a medida que sus cadenas de entrada **w** son más grandes.
- . Por eso el tiempo de M no se mide en términos absolutos sino con funciones temporales **T(n)**, que se definen en términos del tamaño **|w|** de **w** (se usa **n = |w|**).
- . Ejemplos de funciones temporales **T(n)**:
 - 5n + 8, 3n², 3n³ + n² + 25n, etc. polinomiales o poly(**n**) *** **n** como factor.
 - 20^(log₂**n**), 12ⁿ + 10n + 5, 6ⁿ5, etc. exponenciales o exp(**n**)*** **n** como exponente.
 - Otras: 2 elevado a la 2ⁿ (doble exponencial), 2 elevado a la 2 elevado a la 2ⁿ (triple exponencial), etc.
- . Una función **T₁(n)** es del orden de una función **T₂(n)**, que se anota así: **T₁(n) = O(T₂(n))**, si para todo **n >= n₀** se cumple **T₁(n) ≤ c.T₂(n)**, con **c > 0**.

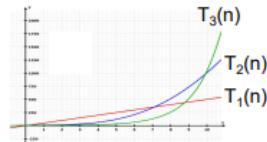
Por ejemplo (**ejercicio**):

$$5n^3 + 8n + 25 = O(n^3)$$

$$n^2 = O(n^3)$$

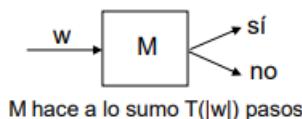
$$n^3 = O(2^n)$$

$$n \log_2 n = O(n^2)$$

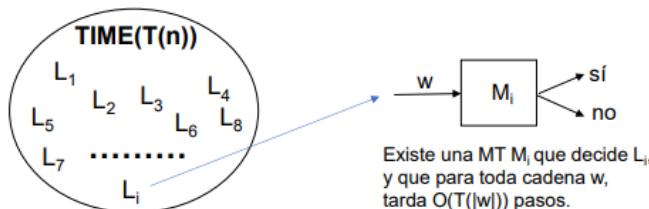


$T_i(n) = O(T_j(n))$, con $i < j$
Cuando n tiende a infinito,
 $T_i(n)$ se mantiene por debajo de $T_j(n)$

- . Usar funciones $O(T(n))$ por funciones $T(n)$ permite manejar un nivel de abstracción adecuado (tiempo lineal, cuadrático, polinomial, cuasipolinomial, subexponencial, exponencial, doble exponencial, etc).
- . Una MT M tarda tiempo $T(n)$, si a partir de toda entrada w , con $|w| = n$, M hace a lo sumo $T(n)$ pasos.



- . Un lenguaje $L \in \text{TIME}(T(n))$ si existe una MT M que lo decide en tiempo $O(T(n))$.

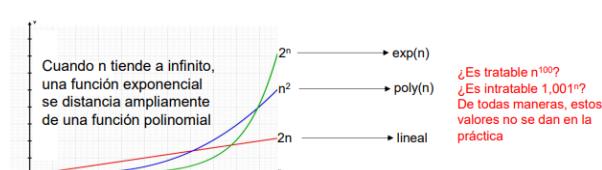


- . Se considera el tiempo máximo (procesar cualquier cadena w consume a lo sumo $T(|w|)$ pasos).

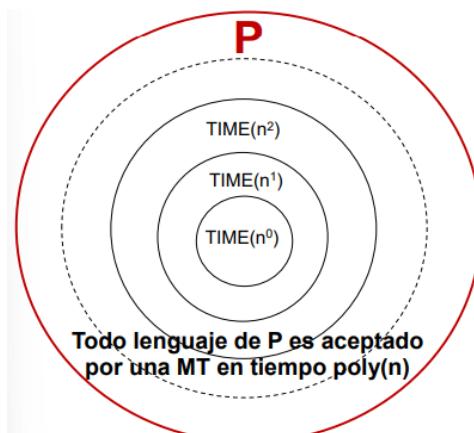
Otros criterios son el tiempo promedio y el tiempo mínimo, en general muy difíciles de calcular (a pesar de buenos avances, al día de hoy se conocen aún pocos valores de este tipo).

- . Convención, respaldada por las matemáticas y la experiencia: tiempo tratable = tiempo $\text{poly}(n)$:

n	$2n$	n^2	2^n
0	0	0	1
1	2	1	2
2	4	4	4
3	6	9	8
4	8	16	16
5	10	25	32
6	12	36	64
7	14	49	128
8	16	64	256
9	18	81	512
10	20	100	1024



LA CLASE P



Robustez: si una MT M_1 con K_1 cintas tarda tiempo $\text{poly}(n)$, existe una MT M_2 equivalente con K_2 cintas que también tarda tiempo $\text{poly}(n)$.

. Ejemplo de lenguaje en la clase P:

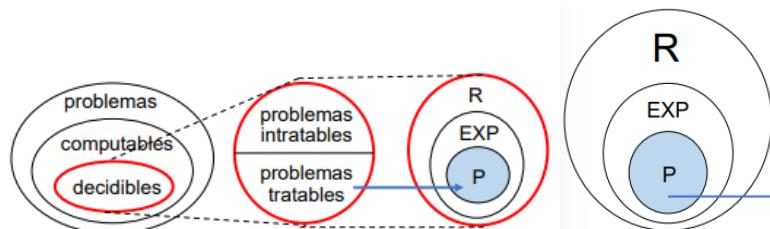
$$L = \{w \mid w \text{ es un palíndromo (capicúa) con símbolos a y b}\}$$

Una MT muy simple que decide L, con 2 cintas, hace:

1. Copia w de la cinta 1 a la cinta 2 Tiempo $O(n)$ ¿por qué? Porque tiene que recorrer los n caracteres y es lineal.
2. Posiciona el cabezal de la cinta 1 a la izquierda Tiempo $O(n)$ ¿por qué? Porque tiene que hacer n pasos para volver al inicio y es lineal.
3. Compara en direcciones contrarias uno a uno los símbolos de las 2 cintas hasta llegar a un blanco en ámbas Tiempo $O(n)$ ¿por qué? Porque en cada cinta el cabezal recorre n caracteres y es lineal.

$$\text{Tiempo total: } T(n) = O(n) + O(n) + O(n) = O(n)$$

PRIMERA VERSIÓN DE LA JERARQUÍA TEMPORAL



R es la clase de los lenguajes recursivos.

P es la clase de los lenguajes decidibles en tiempo $\text{poly}(n)$, es decir tiempo $O(n^k)$.

EXP es la clase de los lenguajes decidibles en tiempo $\exp(n)$, es decir tiempo $O(c^{(\text{poly}(n))})$.

Se prueba que $P \subset EXP \subset R$.

Intratable = Ineficiente.

. *Tesis Fuerte de Church-Turing*:

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

. *Ejemplo de lenguaje que no estaría en P*:

SAT = $\{\varphi \mid \varphi \text{ es una fórmula booleana sin cuantificadores satisfactible con } m \text{ variables}\}$.

P.ej.: $\varphi_1 = (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_3)$ es satisfactible con la asignación $\mathcal{A} = (V, V, V)$

$\varphi_2 = (x_1 \wedge x_2 \wedge x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ no es satisfactible con ninguna asignación

Una MT M que decide SAT emplea una tabla de verdad (no se conoce otro algoritmo). P.ej., para φ_2 :

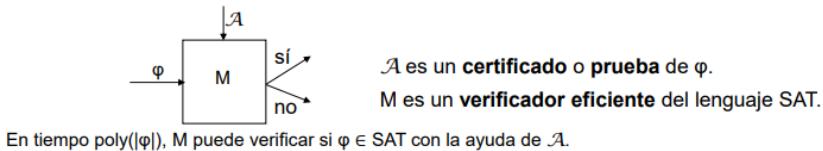
$(x_1 \wedge x_2 \wedge x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$						
V	V	V	F	V	V	V
V	V	F	F	V	V	F
V	F	V	F	V	F	V
V	F	F	F	V	F	F
F	V	V	F	F	V	V
F	V	F	F	V	F	F
F	F	V	F	F	V	
F	F	F	F	F	F	F

] 2^m posibles asignaciones (en este caso 8)
(por cada variable se prueba con los valores V y F)

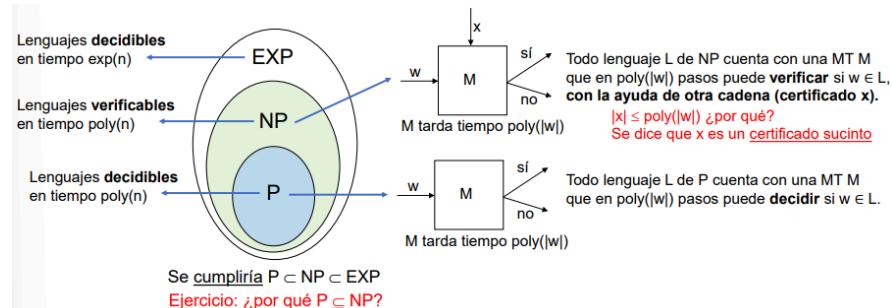
La evaluación de cada asignación
se puede efectuar en tiempo $O(|\varphi_2|^2)$

Por lo tanto, M puede llegar a ejecutar $O(2^m \cdot |\varphi_2|^2) = O(2^n \cdot n^2) = \exp(n)$ pasos.

Dada una fórmula booleana φ , una MT M puede llegar a decidir si $\varphi \in \text{SAT}$ en tiempo exponencial, pero contando con una asignación A, M puede verificar si $\varphi \in \text{SAT}$ en tiempo polinomial.



SEGUNDA VERSIÓN DE LA JERARQUÍA TEMPORAL (ACOTADA A LA CLASE EXP)



- . Si un problema está en P, podemos asegurar que sus soluciones **se encuentran eficientemente**.
- . Si un problema está en NP, sólo podemos asegurar que sus soluciones **se verifican eficientemente**.
- . Intuitivamente, $P \neq NP$ (encontrar una solución parece ser más difícil que verificarla). Sin embargo, al día de hoy esta relación **no ha podido ser probada**.

EL PROBLEMA P vs NP ($P = NP$ o $P \neq NP$?)

- . P = lenguajes que tiene una MT M que los resuelve en tiempo eficiente.
 - . El problema P vs NP fue planteado hace más de 50 años. Es uno de los siete problemas del milenio.
 - . Miles de lenguajes de interés están hoy día en NP – P (lenguajes relacionados con problemas de la lógica, grafos, aritmética, teoría de conjuntos, álgebra, combinatoria, redes, etc).
 - . Pareciera que la única manera de decidir los lenguajes hoy día en NP – P es por medio de la fuerza bruta (búsqueda exhaustiva en el espacio de todas las soluciones posibles).
 - . Por el contrario, los algoritmos que deciden los lenguajes de P son analíticos (parten del conocimiento profundo del problema, y se han elaborado con el empleo de ingenio, creatividad, experiencia, etc).
 - . En la práctica se asume $P \neq NP$, y por eso a los lenguajes de interés hoy día en NP – P se los procesa con remediaciones específicas (cadenas de cierta longitud y/o forma, aproximaciones polinomiales, etc).
- Se cumple $P \subset EXP$ = prueba por diagonalización.
- ¿Se cumple $P \subset NP \subset EXP$? = $P \subset NP$ se cumple ya que de no ser así, la entrada X no se utilizaría en NP y sería lo mismo que P.

OTRAS MÉTRICAS DE COMPLEJIDAD COMPUTACIONAL

. Dinámicas:

Cantidad de cambios de dirección del cabezal de una MT (Hennie).

Consumo de recursos abstractos (Blum).

Etc.

. Estáticas:

Mínimo valor $|Q| \cdot |\Gamma|$ de una MT (Shannon).

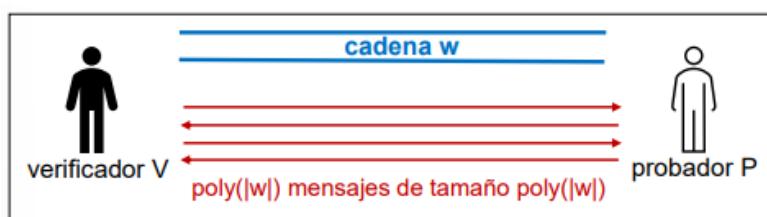
Complejidad estructural de una MT (Chomsky): según el tipo de cintas, el tipo de movimientos, el espacio recorrido, etc. (autómatas finitos, autómatas con pila, autómatas linealmente acotados, MT generales).

Etc.

DEFINICIÓN EQUIVALENTE DE LA CLASE NP

Un probador P (MT de poder ilimitado) tiene que convencer a un verificador V (MT de tiempo $\text{poly}(n)$) que una cadena w pertenece a un lenguaje L .

Para ello, V y P intercambian preguntas y respuestas. Las preguntas y respuestas miden y suman $\text{poly}(|w|)$.



En la definición anterior de NP: P le envía a V UN certificado.

En la definición nueva de NP (equivalente): P intercambia con V varios mensajes.

- Si $L \in \text{NP}$, entonces existe V tal que:

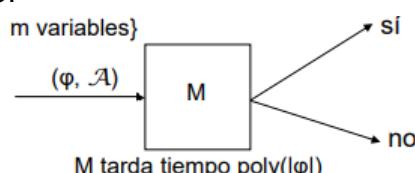
Si $w \in L$, existe un P que puede convencer a V .

Si $w \notin L$, no existe ningún P que pueda convencer a V .

COMPLEMENTOS DE LOS LENGUAJES DE NP

- SAT = $\{\varphi \mid \varphi \text{ es una fórmula booleana satisfactible sin cuantificadores con } m \text{ variables}\}$
SAT no estaría en P: hay 2^m asignaciones A para chequear.

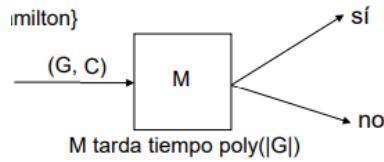
SAT está en NP: dadas una fórmula booleana φ y una asignación A, se puede verificar en tiempo $\text{poly}(n)$ si A satisface φ .



- CH = $\{G : G \text{ es un grafo no dirigido con } m \text{ vértices y tiene un circuito de Hamilton}\}$

CH no estaría en P: hay $m!$ permutaciones C de V para chequear.

CH está en NP: dados un grafo G y una permutación C de V, se puede verificar en tiempo $\text{poly}(n)$ si C es un CdeH de G.

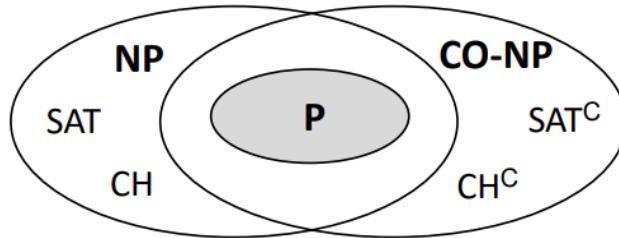


CO-NP tiene los complementos de los lenguajes de NP.

La conjetura aceptada es que $\text{NP} \neq \text{CO-NP}$. Es decir que NP no sería cerrada con respecto al complemento.

En cambio, P sí es cerrada con respecto al complemento (ejercicio), lo que refuerza la conjetura $P \neq \text{NP}$.

Otra conjetura aceptada es que $P \subset \text{NP} \cap \text{CO-NP}$.



COMPLEJIDAD TEMPORAL Y SU RELACIÓN CON LA CODIFICACIÓN DE CADENAS

Ejemplo. Sea $\text{DIV-3} = \{N \mid N \text{ es un número natural que tiene un divisor que termina en } 3\}$

- Una MT M que decide DIV-3 divide N por 3, 13, 23, etc., hasta encontrar eventualmente un divisor de N (no se conoce otro algoritmo).

P. ej., si $N = 100$, M divide N por 3, 13, 23, 33, 43, 53, 63, 73, 83, 93 (10 divisiones).

M ejecuta unas $N/10$ iteraciones, es decir $O(N)$. Cada división se puede hacer en tiempo polinomial.

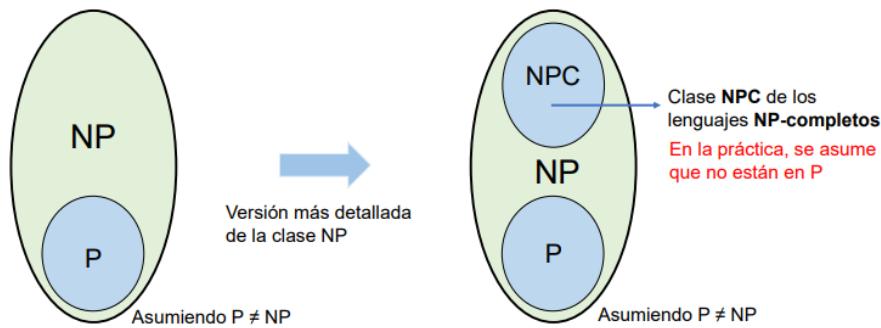
- Por lo tanto, M ejecuta $O(N)$ iteraciones de tiempo polinomial. Resta expresar N en términos de n:

- 1) Si N se codifica en unaryo, $n = N$. Así, M hace $O(n) = \text{poly}(n)$ iteraciones.
- 2) Si N se codifica en binario, $n = O(\log_2 N)$, y por lo tanto $N = O(2^n)$. Así, M hace $\exp(n)$ iteraciones.
- 3) Si N se codifica en cualquier otra base, M hace $\exp(n)$ iteraciones.

Clase 6 | NP-Compleitud

LENGUAJES NP-COMPLETOS

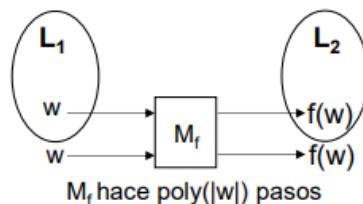
Una manera de reforzar la sospecha de que un lenguaje de NP no está en P es probando que es NP-completo.



Se prueba que si un lenguaje NP-completo está en P, se cumple $P = NP$. Por lo tanto, si se cumple $P \neq NP$, un lenguaje NP-completo no está en P.

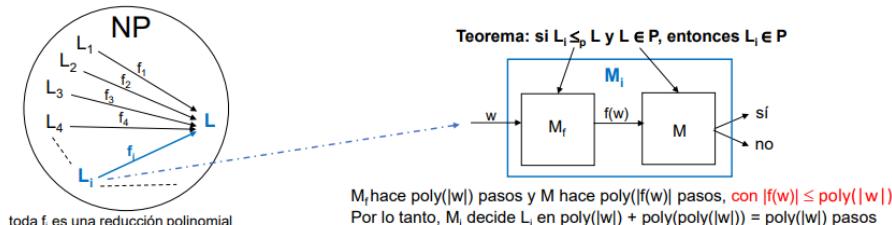
. Definición y Utilidad de la NP-Complejidad:

Una reducción polinomial de un lenguaje L_1 a un lenguaje L_2 es una reducción de L_1 a L_2 de tiempo polinomial ($L_1 \leq_p L_2$).



Un lenguaje L es NP-completo, o $L \in NPC$, si:

1. $L \in NP$
2. Todo $L_i \in NP$ cumple $L_i \leq_p L$ (se dice que L es NP-difícil). Cualquier lenguaje $L \in NP$ se puede reducir a L



Si L está en P, todo L_i de NP está en P, es decir: $NP = P$. Así, si $P \neq NP$, entonces L no está en P.

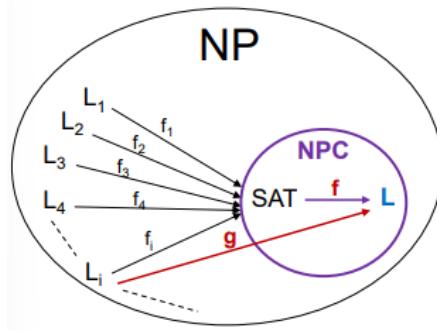
Si una máquina trabaja en tiempo $\text{poly}(|w|)$, la salida no puede exceder ese tiempo.

. Cómo encontrar un lenguaje L que sea NP-Completo:

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

Se cumple que L es NP-completo:

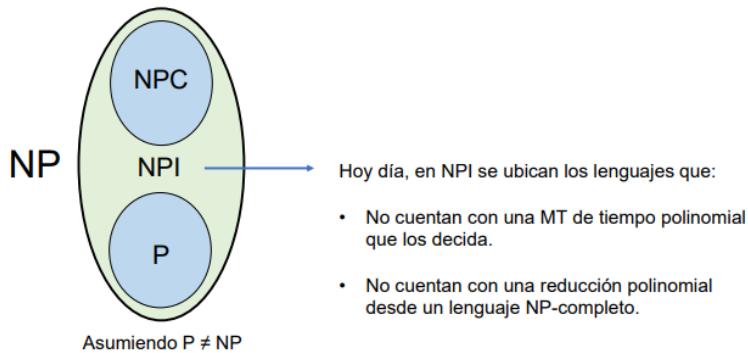
- Sea g la composición de f_i con f .
- La función g es una reducción polinomial de L_i a L . Las reducciones polinomiales son transitivas, es decir que componer 2 reducciones me da otra reducción. Si ambas son polinómicas entonces el resultado también lo será.
- Como lo anterior vale para todo L_i de NP, entonces L es NP-completo.



Todo lenguaje finito es recursivo y está en P.

LA CLASE NPI

Asumiendo $P \neq NP$, entre las clases P y NPC de la clase NP se encuentra la clase NPI, llamada así porque identifica a los lenguajes de NP de dificultad intermedia.



. Problema de Factorización:

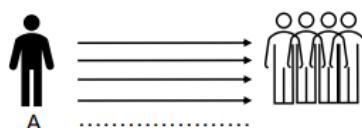
El problema de factorización es candidato a pertenecer a la clase NPI.

Definición: dado N , encontrar sus factores primos. Por ejemplo, $180 = 2^2 \times 3^2 \times 5$.

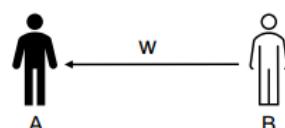
Lenguaje asociado: $\text{FACT} = \{(M, A, B) \mid \text{existe un factor primo de } M \text{ entre } A \text{ y } B\}$.

Asumiendo $\text{FACT} \notin P$, se lo utiliza para implementar un sistema de seguridad de clave pública y privada.

Fundamento: si $N = N_1 \times N_2$ es muy grande, y N_1 y N_2 son números primos, se necesitaría mucho tiempo (tiempo exponencial) para obtener N_1 y N_2 de N :



A permite que todos conozcan la **clave pública** (incluso los **hackers**), basada en **N**. Sólo A tiene la **clave privada**, basada en **N_1 y N_2** .



B le envía a A un mensaje w , encriptado con una función E:
 $v = E(w, N)$

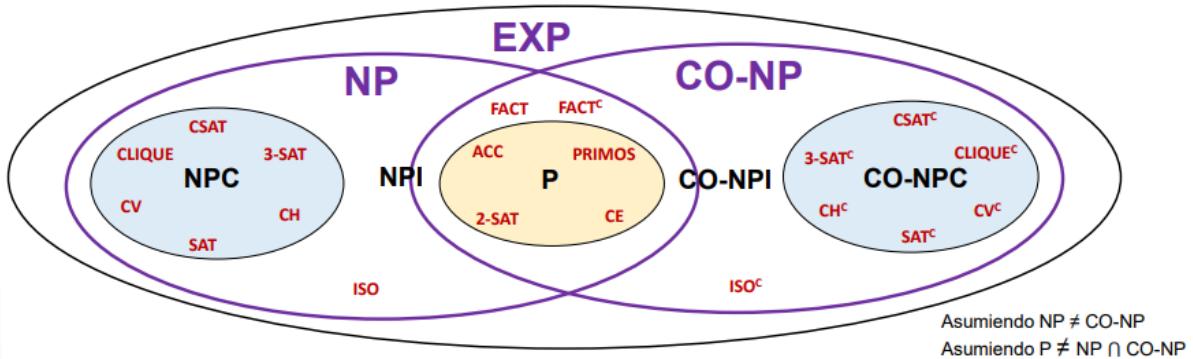
$$v = 11110100001110101110001111101010$$



A desencripta el mensaje w , por medio de una función D:
 $w = D(v, N_1, N_2)$

$$w = 10111100001010111011|101010111$$

ÚLTIMA VERSIÓN DE LA JERARQUÍA TEMPORAL



Se cumple que $NP \neq CO-NP$ implica $P \neq NP$.

Se cumple que NPC y $NP \cap CO-NP$ son disjuntos.

RELACIÓN ENTRE LENGUAJES NP-COMPLETOS

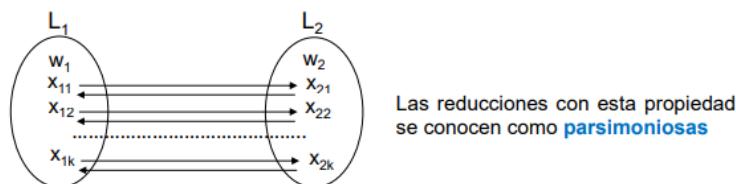
No sólo todo par de lenguajes NP-completos conocidos L_1 y L_2 son p-isomorfos:

$$L_1 \xrightleftharpoons[f]{f^{-1}} L_2 \quad \begin{array}{l} \text{La función } f \text{ y la función inversa } f^{-1} \\ \text{son reducciones polinomiales} \end{array}$$

Sino que también en su mayoría, dos lenguajes NP-completos conocidos L_1 y L_2 cumplen que los certificados de sus cadenas se pueden transformar eficientemente en uno y en el otro sentido,



y más aún, el número de certificados de uno coincide con el número de los certificados del otro:



Las reducciones con esta propiedad se conocen como **parsimoniosas**

P VS NP

La conjetura ampliamente aceptada es que $P \neq NP$, pero no se descartan otras posibilidades.

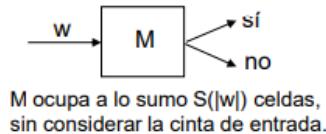
1. Que haya algoritmos eficientes para los lenguajes NP-completos, pero con exponentes muy grandes, del tipo $O(n^{100})$. En un escenario así, el costo para obtener una solución seguiría siendo mucho mayor que el costo para verificarla.
2. Que se demuestre que hay un algoritmo eficiente para un lenguaje NP-completo, pero con una prueba no constructiva. En este caso, aún sabiendo de su existencia, no tendríamos idea alguna sobre la estructura del algoritmo.

3. Que las conjeturas $P \neq NP$ y $P = NP$ no puedan demostrarse con la lógica existente. En esta situación, aun contando con un algoritmo eficiente para un lenguaje NP-completo, no lo podríamos comprobar.

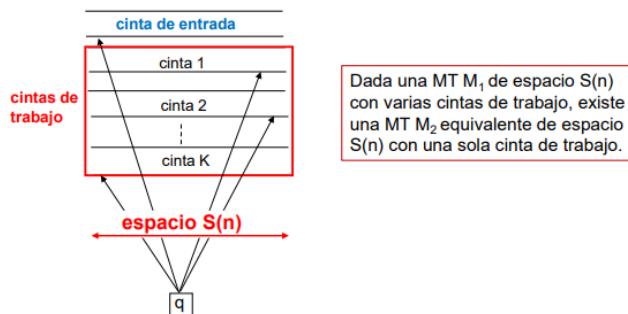
Clase 7 | Jerarquía Espacio-Temporal

COMPLEJIDAD ESPACIAL

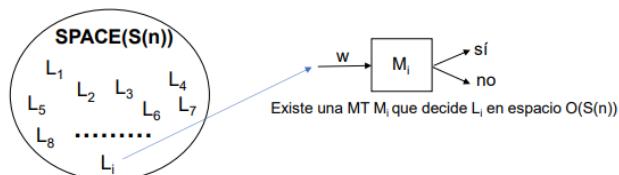
. Una MT M ocupa espacio $S(n)$ si al ejecutarse desde toda entrada w , con $|w| = n$, M ocupa a lo sumo $S(n)$ celdas en cualquier cinta distinta de la de entrada.



. La cinta de entrada es de sólo lectura y no se considera en la medición del espacio. Esto permite un espacio menor que lineal, es decir menor que $O(n)$ (la cadena de entrada mide n).

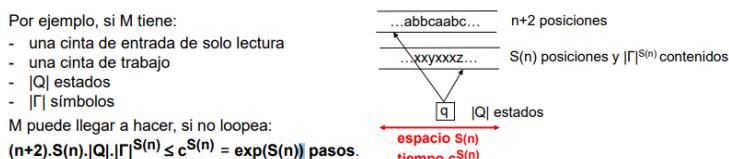


. Un lenguaje pertenece a la clase $SPACE(S(n))$ si existe una MT M que lo decide en espacio $O(S(n))$. Siempre para.



. Una MT M de tiempo $T(n)$ ocupa a lo sumo espacio $T(n)$ celdas. Es decir, si tarda 100, a lo sumo (como mucho) tendrá 100 celdas, pueden ser menos.

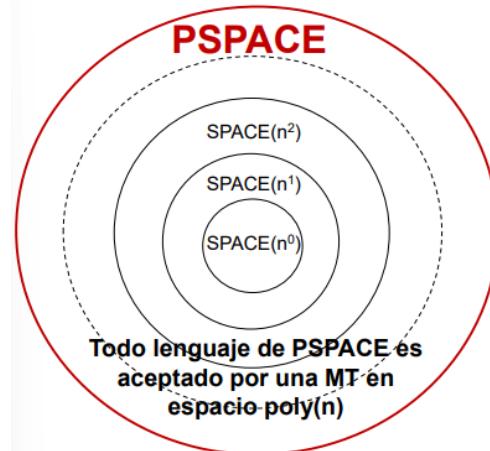
. En cambio, una MT M que ocupa espacio $S(n)$ puede tardar mucho más que tiempo $S(n)$:



$\exp(S(n))$ pasos → tarda en hacer todas las posibles combinaciones.

En un espacio dado, el tiempo es mucho más que ese espacio (todas las combinaciones posibles).

LA CLASE PSPACE



Robustez: si una MT M_1 con K_1 cintas ocupa espacio $\text{poly}(n)$, existe una MT M_2 equivalente con K_2 cintas que también ocupa espacio $\text{poly}(n)$.

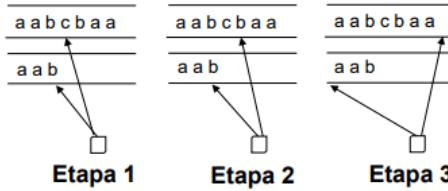
Ejemplo de lenguaje en la clase PSPACE

$$L = \{vcv^R \mid v^R \text{ es la inversa de } v \text{ y tiene símbolos } a \text{ y } b\}$$

La siguiente MT M decide L en espacio $O(n)$:

1. Copia la entrada hasta la c exclusive en la cinta 2.
2. Saltea la c en la cinta 1.
3. Compara símbolo a símbolo las dos cintas, yendo a la derecha en la cinta 1 y a la izquierda en la cinta 2, hasta llegar a un blanco en las dos cintas, en cuyo caso acepta.

Por ejemplo:



Este problema se puede resolver en espacio $O(\log_2 n) \rightarrow$ La clase LOGSPACE.

LA JERARQUÍA ESPACIAL

LOGSPACE es la clase de los lenguajes decidibles en espacio $O(\log_2 n)$.

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

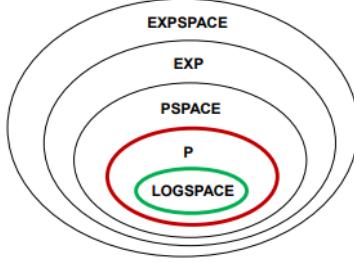
EXPSPACE es la clase de los lenguajes decidibles en espacio $\exp(n)$.

De antes: **espacio $S(n)$ implica tiempo $c^{S(n)}$** , con c constante

En particular: **espacio $\log_2 n$ implica tiempo $c^{\log_2 n}$** , con c constante

Pero: $c^{\log_2 n} = n^{\log_2 c}$, con c constante, es decir **$\text{poly}(n)$**

por lo tanto: **espacio $\log_2 n$ implica tiempo $\text{poly}(n)$**



Se prueba que $\text{LOGSPACE} \subseteq \text{PSPACE}$

Se conjectura que $\text{LOGSPACE} \subseteq \text{P}$

Se conjectura que $\text{P} \subseteq \text{PSPACE}$

En definitiva, $\text{LOGSPACE} \subseteq \text{P}$, es decir que los lenguajes de LOGSPACE son tratables.

. 2 maneras de demostrar:

Verificando solo un fragmento de la entrada:

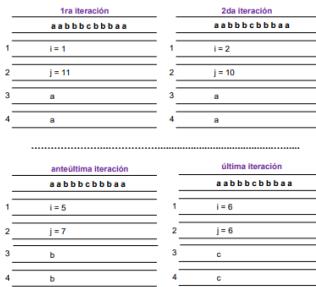
- En realidad, $L = \{v v^R \mid v^R \text{ es la inversa de } v \text{ y tiene símbolos a y b}\}$ se puede decidir en menos espacio:

- La siguiente MT decide L en espacio $O(\log n)$. Veámoslo por ejemplo con la entrada $w = aabbcbbaaa$:

1. Hace $i := 1$ en la cinta 1.
2. Hace $j := n$ en la cinta 2. Si j es par, rechaza.
3. Copia el símbolo i de w en la cinta 3.
4. Copia el símbolo j de w en la cinta 4.
5. Si $i = j$: si los símbolos son **c**, acepta, si no, rechaza.
6. Si $i \neq j$: si los símbolos no son los dos **a** o los dos **b**, rechaza.
7. Hace $i := i + 1$ en la cinta 1.
8. Hace $j := j - 1$ en la cinta 2.
9. Vuelve al paso 3.

La MT M ocupa el espacio de los contadores i y j , que en binario miden $O(\log n)$, más 2 celdas para alojar cada vez a los símbolos comparados (**espacio constante**). Por lo tanto, $L \in \text{SPACE}(\log n)$.

A esta clase se la denomina **LOGSPACE**.

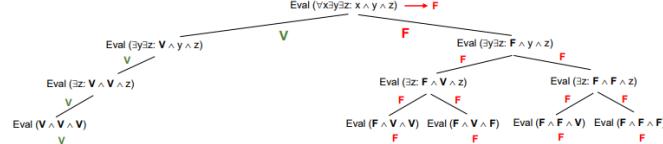


Recursión y uso del espacio con certificados en forma de árboles (no sucintos):

QSAT = { $\varphi \mid \varphi$ es una fórmula booleana con cuantificadores, no tiene variables libres, y es verdadera}.

Por ejemplo: $\varphi_1 = \exists x \exists y \exists z: X \wedge Y \wedge Z$ (fórmula verdadera)
 $\varphi_2 = \forall x \exists y \exists z: X \wedge Y \wedge Z$ (fórmula falsa)

- QSAT está en PSPACE y no estaría en P ni NP. La prueba de que QSAT está en PSPACE se basa en una función recursiva Eval. Por ejemplo:



- El espacio que ocupa una rama de la recursión es **reutilizado** por la rama siguiente.
- La profundidad de la recursión es $O(n)$ (a lo sumo hay tantas invocaciones como símbolos en la fórmula).
- El tamaño de cada instancia invocada es $O(n)$.
- Por lo tanto, el espacio total ocupado es $O(n^2)$.

Notar que ahora los certificados son árboles (no son sucintos)

17

LA JERARQUÍA ESPACIO-TEMPORAL

. QSAT está entre los lenguajes más difíciles de PSPACE. Es PSPACE-completo (todo $L \in \text{PSPACE}$ cumple $L \leq_p \text{QSAT}$).

. Otros problemas PSPACE-completos clásicos se asocian a típicos juegos entre dos jugadores J1 y J2, que plantean decidir si J1 gana: ¿Existe una jugada de J1 tal que para toda jugada de J2 existe una jugada de J1 tal que para toda jugada de J2 ... J1 gana?

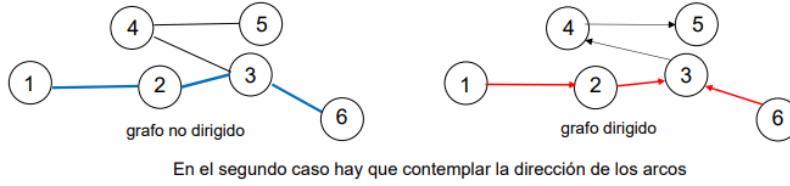
Ejemplos: el ajedrez y el go (asumiendo tableros de $n \times n$ entre otras cosas).

Se plantea un árbol que contemple todas las combinaciones de posibles jugadas a partir de una jugada.

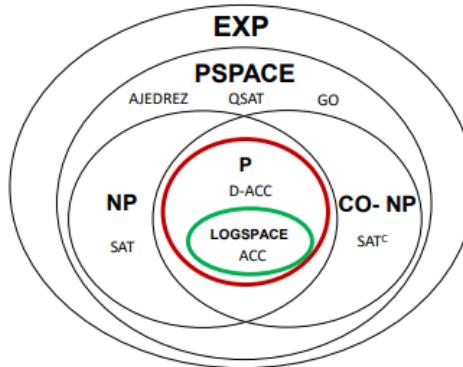
. Por otro lado,

$ACC = \{G \mid G \text{ es un grafo no dirigido con un camino del vértice } 1 \text{ al vértice } m\}$ está en LOGSPACE, y pareciera que el mismo lenguaje pero definido con grafos dirigidos es más difícil:

$D-ACC = \{G \mid G \text{ es un grafo dirigido con un camino del vértice } 1 \text{ al vértice } m\}$ no estaría en LOGSPACE:



En el segundo caso hay que contemplar la dirección de los arcos

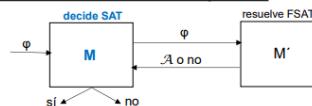


. Complejidad Temporal de los Problemas de Búsqueda:

Las clases P y NP correspondientes a los problemas de búsqueda (o problemas de función) se denominan FP y FNP. Para distinguirlos, a los nombres de los problemas de búsqueda se les antepone una F.

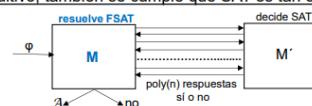
Ejemplo. Problemas FSAT (búsqueda) y SAT (decisión).

Claramente, FSAT es tan o más difícil que SAT:



A partir de M' se puede construir M , tal que:
si M' resuelve FSAT en tiempo $\text{poly}(n)$,
entonces M decide SAT en tiempo $\text{poly}(n)$.
Formalizando:
Si $SAT \in FP$ entonces $SAT \in P$. O lo mismo:
Si $SAT \notin P$ entonces $SAT \notin FP$.

Menos intuitivo, también se cumple que SAT es tan o más difícil que FSAT:



A partir de M' se puede construir M , tal que:
si M' decide SAT en tiempo $\text{poly}(n)$,
entonces M resuelve FSAT en tiempo $\text{poly}(n)$.
Formalizando:
Si $SAT \in P$ entonces $FSAT \in FP$. O lo mismo:
Si $FSAT \notin FP$ entonces $SAT \notin P$.

Esto se cumple para todos los lenguajes NP-completos. También se cumple $P = NP$ si $FP = FNP$.

. Aproximaciones Polinomiales:

Cuando a un problema de búsqueda de óptimo (máximo o mínimo) no se le conoce resolución polinomial, se plantea resolverlo con una aproximación polinomial (MT de tiempo polinomial con una solución "buena").

Ejemplo. Mínimo cubrimiento de vértices de un grafo (no tendría resolución polinomial). La siguiente MT M , en el peor caso, encuentra un cubrimiento con el doble de vértices del mínimo:

Dado $G = (V, E)$, la MT M hace:

1. $V' := \emptyset$ y $E' := E$
2. Si $E' = \emptyset$, acepta
3. $E' := E' - \{(v_1, v_2)\}$, siendo (v_1, v_2) algún arco de E'
4. Si $v_1 \notin V'$ y $v_2 \notin V'$, entonces $V' := V' \cup \{v_1, v_2\}$
5. Vuelve al paso 2

Es decir, se toman arcos no adyacentes de E y se construye un cubrimiento de vértices con sus dos extremos.

El cubrimiento mínimo del grafo de arriba tiene **tamaño 3**. La MT M puede generar, entre otros cubrimientos: $V' = \{1, 2, 3, 5, 4, 6\}$, de **tamaño 6**, y $V' = \{1, 2, 4, 5\}$, de **tamaño 4** (ejercicio: ¿cuáles otros?).

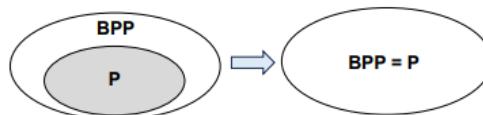
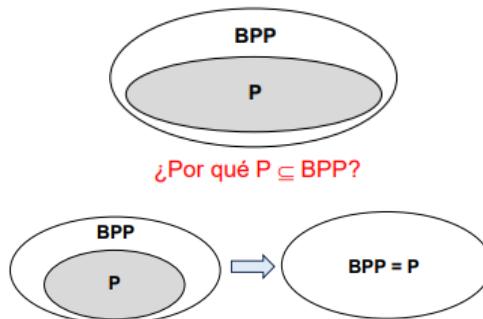
. MT Probabilísticas o MTP (algoritmos aleatorios):

Una MT probabilística (MTP), en cada paso elige aleatoriamente una entre dos continuaciones, cada una con probabilidad $1/2$ ("tiro de moneda").

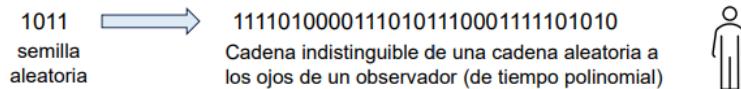
Un lenguaje L pertenece a la clase BPP (bounded probabilistic polynomial) si existe una MTP M con computaciones de tiempo $\text{poly}(n)$ tal que:

- Si $w \in L$, M acepta w en al menos $2/3$ de sus computaciones.
- Si $w \notin L$, M rechaza w en al menos $2/3$ de sus computaciones.

Por lo tanto, las MTP asociadas a BPP tienen una probabilidad de error $\epsilon \leq 1/3$.

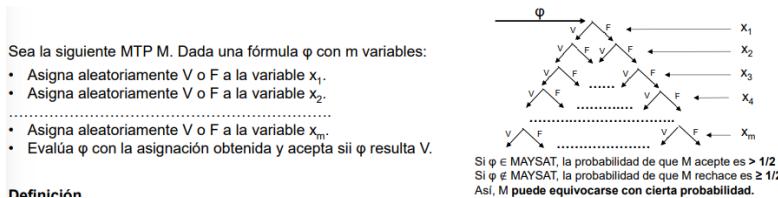


La conjectura más aceptada es que $P = BPP$. Es decir, todo algoritmo probabilístico polinomial se podría desaleatorizar con un retardo sólo polinomial, por la posibilidad de simular lo aleatorio con generadores pseudoaleatorios, que generan a partir de pequeñas cadenas aleatorias (semillas) grandes cadenas que "parecen" aleatorias:



Ejemplo 1:

$MAYSAT = \{\varphi \mid \varphi$ es una fórmula booleana satisfactible por al menos la mitad más uno de las asignaciones}



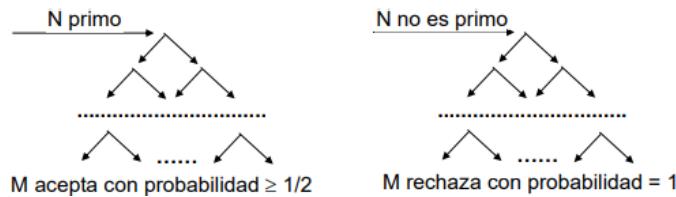
Ejemplo 2:

$PRIMOS = \{N \mid N$ es un número primo}

A diferencia de MAYSAT, PRIMOS pertenece a BPP:

Existe una MTP M que, dada una entrada N:

- Si $N \in PRIMOS$, M acepta en al menos la mitad de sus computaciones.
- Si $N \notin PRIMOS$, M rechaza en todas sus computaciones.



Es decir:

Si N no es primo, M nunca se equivoca.

Si N es primo, M se puede equivocar con probabilidad a lo sumo $1/2$.

De todos modos, la probabilidad de error de M se puede decrementar significativamente:

- Ejecutando M dos veces, la probabilidad de que se equivoque es a lo sumo $1/4$.
- Ejecutando M tres veces, la probabilidad de que se equivoque es a lo sumo $1/8$.
- En general, después de k ejecuciones, la probabilidad de que M se equivoque es a lo sumo $1/2^k$.

. Máquinas Cuánticas o MC (algoritmos cuánticos):

Las máquinas cuánticas (MC), a diferencia de las máquinas clásicas, tienen cubits en lugar de bits.

Un cúbit, como un bit, puede estar en los estados básicos 0 o 1, pero a diferencia de un bit, puede estar también en un estado de superposición de 0 y 1. Los estados de un cúbit se suelen anotar así:

Estados posibles de un cúbit

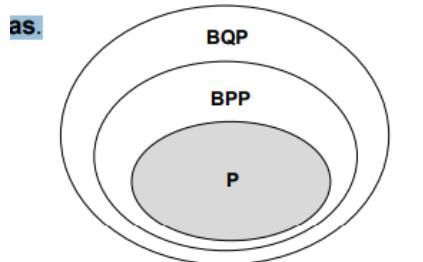
$$|0\rangle \quad |1\rangle \quad \alpha_0|0\rangle + \alpha_1|1\rangle$$

En el último caso, α_0 y α_1 son números complejos (amplitudes) que cumplen $|\alpha_0|^2 + |\alpha_1|^2 = 1$, tal que:

- El cúbit está en un estado de superposición de 0 y 1.
- Al medirlo (leerlo), se obtiene el valor 0 con probabilidad $|\alpha_0|^2$ o el valor 1 con probabilidad $|\alpha_1|^2$. Una vez que lo leo rompo la superposición.
- Luego de la medición se destruye la superposición (el cúbit queda en el estado básico obtenido, 0 o 1).

Los cambios en los cubits se hacen con operaciones o puertas cuánticas. Una puerta cuántica se aplica sobre uno, dos o tres cubits. La cantidad de puertas ejecutadas establece el tiempo de la MC.

La clase BQP (bounded-error quantum polynomial time), contiene los lenguajes aceptados por MC de tiempo $\text{poly}(n)$, con probabilidad de error $\leq 1/3$. Como las MTP, las MC pueden equivocarse, pero la probabilidad de error, repitiendo ejecuciones, es muy baja. BQP es la clase P cuántica.



Por lo tanto, las MC podrían refutar la
Tesis Fuerte de Church-Turing

Se cumple $P \subseteq BQP$ y también $BPP \subseteq BQP$. La conjectura más aceptada es que las dos inclusiones son estrictas.

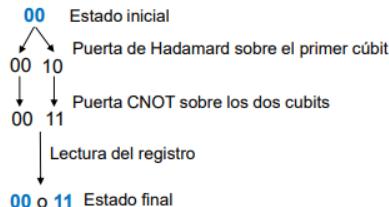
Otra conjetura aceptada es que BQP y NP son incomparables (ninguna clase incluye a la otra). También se cumple $BQP \subseteq PSPACE$.

Las MC no podrían decidir eficientemente los lenguajes NP-completos.

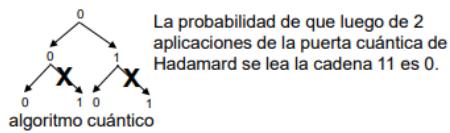
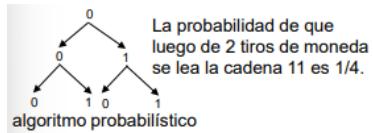
Ejemplo de Puertas Cuánticas y Computación Cuántica

Puerta de Hadamard: dado un cúbbit en estado básico 0 o 1, lo pasa al estado de superposición 0 y 1.

Puerta CNOT: dados dos cubits, invierte el estado del 2do sólo si el 1ro es 1.

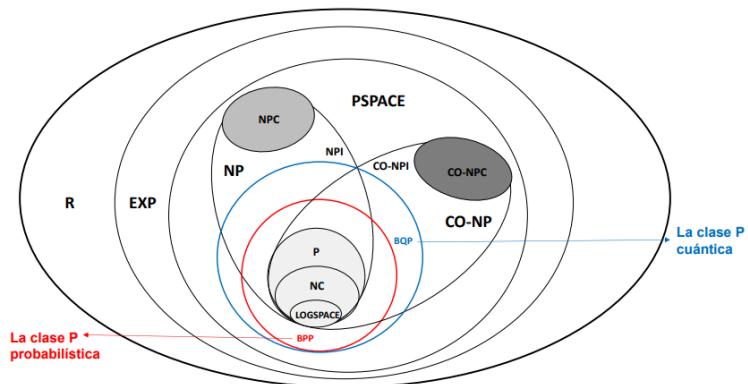


Diferencia entre algoritmos Probabilísticos y algoritmos Cuánticos



Sólo en las MC las computaciones **se interfieren**.
Esto marcaría su **real ventaja** sobre las MTP.

JERARQUÍAS



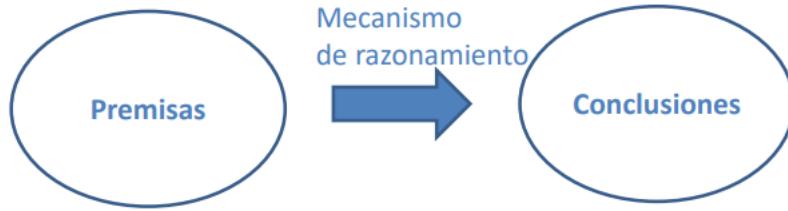
Clase 8 | Lógica de Enunciados o Proposicional

LÓGICA

. Trata acerca de los medios a través de los cuales puede propagarse y articularse el conocimiento. Es la disciplina que estudia las formas de razonamiento.

RAZONAMIENTO

. El razonamiento es el que nos lleva de premisas a conclusiones. Podemos decir que es un proceso de construcción de nuevo conocimiento.



. El razonamiento se apoya en verdades supuestas, para obtener otras como resultado de la actividad de razonamiento. Es decir que a partir de premisas, verifico que estas sean verdaderas para obtener nuevas verdades (conclusiones).

. Sirve para justificar ciertas verdades, si son consecuencia (formal) de conocimientos previamente aceptados:

- Premisas del razonamiento (propuestas).
- Conclusión (conocimiento nuevo).

Los conocimientos se expresan mediante proposiciones o enunciados (lenguaje de la lógica).

Razonamiento: encadenamiento de enunciados formado por premisas y conclusiones.

. El razonamiento puede ser:

- correcto o válido: si la manera en que está construido garantiza la conservación de la verdad. Si las premisas son V, entonces la conclusión es necesariamente V.
- Incorrecto o inválido: su construcción es defectuosa (no hay garantía acerca del valor de verdad de la conclusión).

. Deducción = Razonamiento Correcto.

. Un razonamiento es directamente incorrecto cuando a partir de premisas verdaderas permite arribar a una conclusión falsa. O también puede ser incorrecto porque tiene la estructura de un razonamiento incorrecto (aunque la conclusión sea verdadera).

. La corrección de la forma solamente garantiza que si las premisas son verdaderas entonces lo será también la conclusión.

ARGUMENTACIONES

. Una forma argumentativa es una sucesión finita de formas enunciativas, de las cuales la última se considera como la conclusión de las anteriores, conocidas como premisas. La notación es:

$$A_1, A_2, \dots, A_n \therefore A$$

. Para que una forma argumentativa sea válida debe representar un razonamiento correcto. Es decir, bajo cualquier asignación de valores de verdad a las variables de enunciado, si las premisas A_1, A_2, \dots, A_n , toman el valor V, la conclusión A también debe tomar el valor V.

. Una forma argumentativa $A_1, A_2, \dots, A_n \therefore A$ es inválida si es posible asignar valores de verdad a las variables de enunciado que aparecen en ella, de tal manera que A_1, A_2, \dots, A_n , tomen el valor V y A tome el valor F. De lo contrario la forma argumentativa es válida.

EL LENGUAJE DE LA LÓGICA

La lógica proposicional, también conocida como lógica de enunciados, es un sistema formal cuyos elementos representan proposiciones o enunciados. Esta lógica no tiene, por sí misma, mucha utilidad para la representación del conocimiento. Está justificado detenerse

en ella porque permite introducir de una manera sencilla algunos conceptos que, explicados directamente para la lógica de predicados, son más difíciles de captar.

En el lenguaje de la lógica hablaremos de enunciados o proposiciones para referirnos a frases, y distinguiremos entre enunciados atómicos (simples) o enunciados compuestos.

Para construir estos segundos, utilizaremos conectivos:

$\neg A$	Negación de A
$A \wedge B$	Conjunción de A y B
$A \vee B$	Disyunción de A o B
$A \rightarrow B$	Si A entonces B
$A \leftrightarrow B$	A si y sólo si B

. *El lenguaje simbólico de la lógica:*

Debemos definir SINTAXIS y SEMÁNTICA. Ya que necesitamos:

- Capturar y formalizar las estructuras del lenguaje natural en un lenguaje simbólico, para luego formalizar los mecanismos de razonamiento que se aplican sobre dichas estructuras lingüísticas.

. *Sintaxis:*

Son el conjunto de símbolos primitivos que forman el alfabeto o vocabulario, y que es acompañado por un conjunto de reglas de formación (la gramática) que nos dice cómo construir fórmulas bien formadas a partir de los símbolos primitivos.

Alfabeto

El alfabeto de un sistema formal es el conjunto de símbolos que pertenecen al lenguaje del sistema. Si L es el nombre del sistema de lógica proposicional, entonces el alfabeto de L consiste en:

Una cantidad finita pero arbitrariamente grande de variables proposicionales (o variables de enunciado).

Un conjunto de operadores lógicos o conectivas: \neg , \wedge , \vee , \rightarrow , \leftrightarrow .

Dos signos de puntuación: el paréntesis izquierdo y el paréntesis derecho.

Gramática

La gramática consiste en un conjunto de reglas que definen recursivamente las cadenas de caracteres que pertenecen al lenguaje. A las cadenas de caracteres construidas según estas reglas se las llama fórmulas bien formadas (fbf), y también se las conoce como formas enunciativas.

Las reglas del sistema L son :

- Las variables de enunciado del alfabeto de L son formas enunciativas. Es decir, p,q,...
- Si A y B son formas enunciativas de L, entonces también lo son ($\neg A$), ($A \wedge B$), ($A \vee B$), ($A \rightarrow B$) y ($A \leftrightarrow B$).
- Sólo las expresiones que pueden ser generadas mediante las cláusulas i y ii en un número finito de pasos son formas enunciativas de L.

. *Semántica:*

Se parte desde un enunciado sintácticamente correcto.

La semántica se desenvuelve en el mundo de la interpretación y la satisfacción → *significado*.

La semántica es composicional, ya que más allá de los valores individuales de los enunciados, el valor de un enunciado compuesto estará determinado por el significado de sus conectivos.

La verdad o falsedad de un enunciado compuesto depende de la verdad o falsedad de los enunciados simples que lo constituyen, y de la forma en que están conectados.

Condicional ($A \rightarrow B$):

- “B pasa si A”
- “B si A”
- “B implica A”

Bicondicional ($A \leftrightarrow B$):

- “Ocurre A sólo si B”

Clase 9 | Lógica de Enunciados o Proposicional

INTERPRETACIÓN Y SATISFACCIÓN

. Una **interpretación** es una función que relaciona los elementos de los dominios sintáctico y semántico de la lógica considerada.



. En el caso particular de la lógica proposicional, una interpretación I consiste en una función de valuación v que asigna a cada variable de enunciado el valor de verdad V o F.

Siendo $P = \{p_1, p_2, \dots\}$ el conjunto de variables de enunciado:

$$v : P \rightarrow \{V, F\}$$

. Las funciones de valoración sólo dan valor a las letras, no a las funciones. Sabiendo cual es el valor de P y de Q podemos obtener el valor de la función.

. Para extender el dominio de la función de valuación de las variables de enunciado a las fbf's en general, se define una regla semántica para cada una de las reglas sintácticas de la gramática:

$\models \rightarrow$ “esta fórmula se satisface...” o “esta fórmula se hace verdadera...”

$$\models v p \text{ si y sólo si } v(p) = V$$

$$\models v (\neg A) \text{ si y sólo si no es el caso que } \models v A$$

$$\models v (A \wedge B) \text{ si y sólo si o bien } \models v A \text{ o bien } \models v B \text{ o ambos}$$

$$\models v (A \vee B) \text{ si y sólo si } \models v A \text{ y } \models v B$$

$$\models v (A \rightarrow B) \text{ si y sólo si no es el caso que } \models v A \text{ y no } \models v B$$

$$\models v (A \wedge B) \text{ si y sólo si } \models v (A \rightarrow B) \text{ y } \models v (B \rightarrow A)$$

También lo puedo escribir así: $v(A \wedge B) = V$ si y sólo si $v(A) = V$ y $v(B) = V$.

TAUTOLOGÍA Y CONTRADICCIÓN

. Una **tautología** toma siempre el valor de verdad V, considerando todas y cada una de las posibles asignaciones de valores de verdad a las variables de enunciado que contiene.

Ejemplo: $(p \vee (\neg p))$.

. Si en cambio siempre toma el valor de verdad F, la forma enunciativa se conoce como **contradicción**.

Ejemplo: $(p \wedge (\neg p))$.

EQUIVALENCIAS LÓGICAS

- . Filas: 2^n con n letras.
- . Tablas diferentes: 2^{2^n} con n letras. corresponden a las 2^{2^n} maneras posibles de disponer los valores V y F en la última columna de una tabla de verdad de 2^n filas.
- . Cada semántica la puedo representar de diferentes maneras. Cada semántica admite diferentes formas sintácticas.

Es por esto que podemos construir infinitas formas enunciativas.

. Implicación Lógica:

Sean A y B dos enunciados. Diremos que “A es lógicamente equivalente a B” (lo denotaremos con $A \Leftrightarrow B$) si la forma enunciativa $A \leftrightarrow B$ es una tautología.

Por ejemplo: $\neg(p \vee q)$ es lógicamente equivalente a $((\neg p) \wedge (\neg q))$

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

También Diremos que “A implica lógicamente a B” o que “B es consecuencia lógica de A” (lo denotaremos con $A \Rightarrow B$) si la forma enunciativa $A \rightarrow B$ es una tautología.

Por ejemplo: $(p \wedge q)$ implica lógicamente a p. Si valen $(p \wedge q)$ entonces vale p.

Debemos hacer la tabla de verdad de $A \rightarrow B$ y probar que es una tautología.

. Ejemplos de Equivalencias Conocidas:

- Ley de Doble Negación $\neg(\neg p) \Leftrightarrow p$
- Ley Comutativa de la Conjunción $p \wedge q \Leftrightarrow q \wedge p$
- Ley Comutativa de la Disyunción $p \vee q \Leftrightarrow q \vee p$
- Ley Asociativa de la Conjunción $(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$
- Ley Asociativa de la Disyunción $(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$
- Leyes de De Morgan $\neg(p \vee q) \Leftrightarrow (\neg p) \wedge (\neg q)$
- $\neg(p \wedge q) \Leftrightarrow (\neg p) \vee (\neg q)$
- Leyes de Distribución $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$
- $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$
- Leyes de Absorción $p \wedge p \Leftrightarrow p$
- $p \vee p \Leftrightarrow p$

CONJUNTOS ADECUADOS DE CONECTIVAS

- . Con los conectivos binarios que tenemos actualmente nos alcanza para representar las 2^n posibles valores de las fórmulas que se pueden formar con n letras. Si hubiera al menos uno más, esa columna se repetiría con otra.
- . Un conjunto adecuado de conectivas es un conjunto tal que toda función de verdad puede representarse por medio de una forma enunciativa en la que sólo aparezcan conectivas de dicho conjunto.
- . Demostración: Proposición. Los pares $\{\neg, \wedge\}$, $\{\neg, \vee\}$ y $\{\neg, \rightarrow\}$ son conjuntos adecuados de conectivas.

Armo la tabla de verdad de la fórmula que quiero, tomo las filas en que es verdadera y armo una fórmula utilizando símbolos \wedge . Luego conecto todas las fórmulas con símbolos v. De esta manera me queda algo en la forma normal conjuntiva.

FORMAS NORMALES

Hemos visto que a partir de toda forma enunciativa puede construirse una tabla de verdad. Vamos a formular ahora un resultado en un sentido recíproco:

Proposición. Toda función de verdad es la función de verdad determinada por una forma enunciativa restringida. Llamamos forma enunciativa restringida a una forma enunciativa en la que solamente figuran las conectivas \neg , \wedge , \vee .

Corolario. Toda forma enunciativa, que no es una contradicción, es lógicamente equivalente a una forma enunciativa restringida de la forma:

$$\bigvee_{i=1}^m \bigwedge_{j=1}^n Q_{ij}$$

donde cada Q_{ij} es una variable de enunciado o la negación de una variable de enunciado. Esta forma se denomina forma normal disyuntiva.

. Argumentaciones:

Proposición. La forma argumentativa $A_1, A_2, \dots, A_n \therefore A$ es válida si y sólo si la forma enunciativa $(A_1 \wedge A_2 \wedge \dots \wedge A_n) \rightarrow A$ es una tautología (es decir, si y sólo si la conjunción de las premisas implican lógicamente a la conclusión).

Para referirnos a formas argumentativas válidas utilizamos la siguiente notación:

$\Gamma \models A$

que se lee: “ Γ implica lógicamente a A ” o “ A se deduce de Γ ”, siendo Γ un conjunto de premisas. $= \{A_1, A_2, \dots, A_n\}$

DEMOSTRACIONES

. Contra-Ejemplo:

Alcanza con encontrar un solo ejemplo que no cumpla con la propiedad propuesta para poder afirmar que la propiedad no se cumple.

. Por absurdo:

Consiste en asumir lo contrario a lo que se quiere demostrar \rightarrow

Asumimos lo contrario a lo que queremos demostrar:
Las hipótesis son Verdaderas y la Conclusión es Falsa, o sea:

Hipótesis: A es tautología (i)

$(A \rightarrow B)$ es tautología (ii)

Conclusion falsa: B no es tautología (iii)

Demostración:

(iv) Por (iii), existe una valuación v tal que $v(B)=F$

(v) Por (i), $v(A)=V$

- (vi) Por (ii), $v((A \rightarrow B))=V$
- (vii) Por definición de valoración y por (v) y (vi), $v(B)=V$
- (viii) Por (iv) y (vii) $\rightarrow v(B)=F$ y $v(B)=V \rightarrow$ Absurdo!

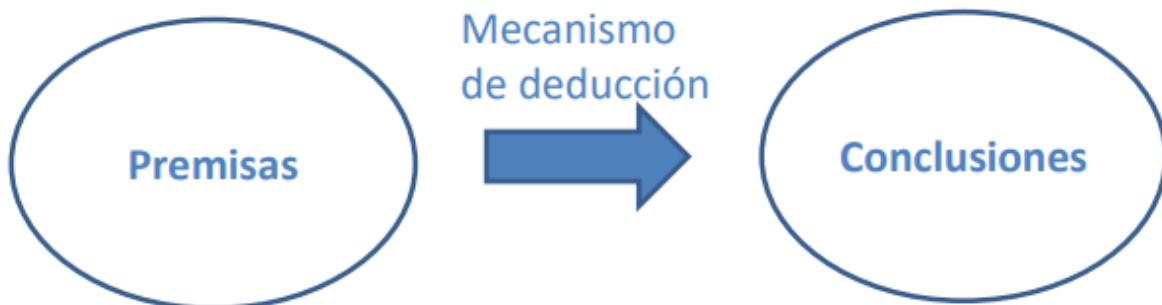
. *Por Inducción Natura/Estructural:*

Se debe demostrar que vale para una unidad y su siguiente, por lo que en el caso de ese siguiente también será válido para su siguiente y así sucesivamente.

Clase 10 | Cálculo de Enunciados L

MECANISMOS FORMALES DE RAZONAMIENTO

Un mecanismo formal de razonamiento (o de inferencia, deducción, demostración) consiste en una colección de reglas que pueden ser aplicadas sobre cierta información inicial para derivar información adicional, en una forma puramente sintáctica.



A continuación se presenta un sistema deductivo llamado L.

EL SISTEMA FORMAL L

Un sistema deductivo axiomático está compuesto por:

- un conjunto de axiomas (en realidad esquemas de axiomas).
- un conjunto de reglas de inferencia.

Los axiomas son fórmulas bien formadas. Las reglas determinan qué fórmulas pueden inferirse a partir de qué fórmulas.

. *Axiomas de L:*

Los axiomas de un sistema axiomático son un conjunto de fórmulas que se toman como punto de partida para las demostraciones.

$$\begin{aligned}
 & (A \rightarrow (B \rightarrow A)) \\
 & (p \rightarrow (p \rightarrow p)) \text{ con } A=p \text{ y } B=p \\
 & ((p \rightarrow p) \rightarrow (q \rightarrow (p \rightarrow p))) \text{ con } A=(p \rightarrow p) \text{ y } B=q
 \end{aligned}$$

. *Reglas de Inferencia de L:*

El sistema L tiene una única regla de inferencia, el modus ponens:

MP: a partir de A y de $(A \rightarrow B)$ se infiere B

Una regla de inferencia es una función que asigna una fórmula (conclusión) a un conjunto de fórmulas (premisas).

DEMOSTRACIÓN O DERIVACIÓN EN L

Una demostración es una sucesión de aplicaciones de reglas de inferencia que permite llegar a una conclusión a partir de determinadas premisas y axiomas.

Como la derivación es una relación transitiva, todas las fórmulas que se vayan obteniendo sucesivamente están implicadas por las premisas o axiomas.



La notación es la siguiente $\Gamma \vdash_L A_n$

Se lee: "A partir de las premisas contenidas en el conjunto Γ (Gamma) se deduce (o infiere o deriva) la conclusión A_n ".

- Γ (gamma) es un conjunto de premisas (o hipótesis o fbf).
- A_n es la conclusión a la que se quiere llegar.

Tanto las premisas como la conclusión son fórmulas bien formadas (fbf).

Para demostrar que la conclusión se obtiene a partir de las premisas, podemos construir una demostración que es una sucesión finita de fórmulas bien formadas A_1, A_2, \dots, A_n , tal que para todo i , con $1 \leq i \leq n$, A_i es una premisa o es un axioma, o bien se infiere de miembros anteriores de la sucesión como consecuencia directa de la aplicación de una regla de inferencia (MP).

TEOREMA DE L

Cuando una fbf se puede demostrar a partir del conjunto vacío de hipótesis se la llama TEOREMA de L.

Por ejemplo $(p \rightarrow p)$ es un teorema de L. La notación es $\vdash_L (p \rightarrow p)$. Gamma no aparece porque está vacío.

Para demostrar que $(p \rightarrow p)$ es un teorema de L tenemos que construir la cadena que termine en $(p \rightarrow p)$, por ejemplo la siguiente cadena es una demostración:

1. $(p \rightarrow ((p \rightarrow p) \rightarrow p)) \rightarrow ((p \rightarrow (p \rightarrow p)) \rightarrow (p \rightarrow p))$ instanciando el axioma L_2
2. $p \rightarrow ((p \rightarrow p) \rightarrow p)$ instanciando el axioma L_1
3. $(p \rightarrow (p \rightarrow p)) \rightarrow (p \rightarrow p)$ MP entre 1 y 2
4. $p \rightarrow (p \rightarrow p)$ instanciando el axioma L_1
5. $(p \rightarrow p)$ MP entre 3 y 4

METATEOREMA DE LA DEDUCCIÓN

Es una herramienta que facilita de gran manera las pruebas.

Si $\Gamma \cup \{A\} \vdash_L B$ entonces $\Gamma \vdash_L (A \rightarrow B)$

Hemos demostrado $\vdash_L (p \rightarrow p)$ para Γ conjunto vacío. Pero nos llevó 5 pasos difíciles.

Aplicando el Metateorema de la deducción lo podemos resolver más fácil:

Si demuestro $\Gamma \cup \{p\} \vdash_L p$ entonces por el MT de la Deducción obtengo $\Gamma \vdash_L (p \rightarrow p)$.

Tengo que demostrar $\Gamma \vdash_L p$ con $\Gamma = \emptyset$.

Esto me lleva un solo paso porque tengo a p de ambos lados.

CORRECCIÓN (Sensatez) Y COMPLETITUD DE UN SISTEMA DEDUCTIVO

De un sistema deductivo se espera naturalmente que:

- Sus demostraciones produzcan conclusiones correctas, es decir que sea una tautología (que sea verdad).
- Y también en lo posible la propiedad inversa, es decir que sea capaz de demostrar todas y cada una de ellas.

Notemos las diferencias entre $\vdash_L A$ y $\models A$

$\vdash_L A$

A es teorema, lo demuestro construyendo su demostración (una secuencia A_1, A_2, \dots, A , tal que para todo i , con $1 \leq i \leq n$, A_i es un axioma, o bien se infiere de miembros anteriores de la sucesión por MP)

Es un procedimiento sintáctico, hago «cuentas» en el cálculo.

$\models A$

A es tautología, lo demuestro construyendo su tabla de verdad y comprobando que da Verdadero en todas las filas.

Es un procedimiento «semántico», aplico las funciones de verdad.

CORRECCIÓN DEL SISTEMA DEDUCTIVO

La corrección de un sistema deductivo está garantizada si se cuenta con axiomas verdaderos y reglas de inferencia correctas (o sensatas), es decir que preservan la verdad. ¿Cómo demostramos que L es correcto? Debemos demostrar lo siguiente:

- Sus axiomas L1, L2 y L3 son tautologías:

Les hacemos la tabla de verdad, por ej:

A	B	$A \rightarrow (B \rightarrow A)$
V	V	V
V	F	V
F	V	V
F	F	V

- Su regla MP preserva la verdad:

Lo demostraremos por el absurdo: Suponemos que MP no preserva la verdad, es decir que a partir de A y de $(A \rightarrow B)$, siendo ambas tautologías, se ha inferido B la cual NO es tautología. Entonces como B no es tautología, existe una valuación, sea v, para las letras de enunciado que aparecen en B (es decir, una fila en su tabla de verdad) para la cual $v(B)=F$. Por otra parte como A es tautología, en esa misma valuación toma el valor Verdadero, o sea $v(A)=V$. ¿Qué valor toma $v(A \rightarrow B)$? Toma el valor F, lo cual contradice la hipótesis que dice que $(A \rightarrow B)$ es tautología. PQD.

Hemos demostrado que L es sensato o correcto:

En símbolos: Si $\vdash_L A$ entonces $\models A$.

En palabras: Si A es teorema de L entonces A es tautología.

PROPIEDADES

. Consistencia:

Un conjunto de Fbf, sea G, no es consistente cuando existe alguna Fbf A tal que:

$G \vdash_L A$ y $G \vdash_L (\sim A)$

Es decir, G permite deducir un enunciado y su negación.

Esta definición se aplica también a L, considerando a G como su conjunto de axiomas.

La propiedad dice que L es consistente.

Si el conjunto de premisas no es consistente, todo se vuelve demostrable a partir de dicho conjunto. En símbolos, sea G un conjunto no consistente, entonces para cualquier Fbf A ocurre que $G \vdash_L A$.

. Decidibilidad:

En la teoría de la computación, se define un problema de decisión como aquél que tiene dos respuestas posibles: "sí" o "no".

En el marco de los sistemas deductivos es muy relevante también la cuestión de la decidibilidad, que se formula de la siguiente manera:

Dados Γ y A, ¿existe algún algoritmo que responda "sí" en el caso de que $\Gamma \models A$ y que responda "no" en el caso de que $\Gamma \not\models A$?

Claramente, en la lógica proposicional esta cuestión tiene una respuesta positiva, utilizando algoritmos basados en las tablas de verdad.

Clase 11 | Lógica de Predicados de Primer Orden

. La lógica proposicional permite formalizar y teorizar sobre la validez de una gran cantidad de enunciados. Sin embargo existen enunciados intuitivamente válidos que no pueden ser probados por dicha lógica.

Por ejemplo, considérese el siguiente razonamiento:

Su formalización es la siguiente:

Todos los hombres son mortales.

p

Sócrates es un hombre.

q

Por lo tanto, Sócrates es mortal.

Por lo tanto, r

Esta es claramente una forma de razonamiento inválido, lo que contradice nuestra intuición. Sucede que para estudiar la validez de este tipo de razonamientos necesitamos analizar:

- ✓ la naturaleza de la premisa «Todos los As son Bs».
- ✓ la estructura interna de cada enunciado.

Dicha problemática la resuelve la [lógica de predicados](#)
Esto se corresponde a las ideas de:

- ✓ [Cuantificadores](#)
- ✓ [Sujetos y Predicados](#)

. El **Sujeto** es la cosa acerca de la cual el enunciado está afirmando algo, ej. Sócrates.

. El **Predicado** se refiere a una propiedad que posee el sujeto, ej «es mortal».

PREDICADOS

- . Un predicado es “lo que se afirma de un sujeto en una proposición”.
- . Los predicados pueden definir propiedades sobre uno, dos, o más individuos (u objetos), establecen relaciones entre ellos.
- . Así, hay predicados unarios (o monádicos), binarios, ternarios, ..., n-arios.
- . Los *predicados unarios definen relaciones de grado uno*, es decir propiedades de un objeto, como por ejemplo:

“el 7 es un número primo” que lo simbolizamos $P_1^1(7)$
«Socrates es mortal» que lo simbolizamos $P_2^1(\text{socrates})$
«Sócrates es un hombre» que lo simbolizamos $P_3^1(\text{socrates})$
«ftc es una materia de la lic en Sistemas» que lo simbolizamos $P_4^1(\text{ftc})$

- . Los predicado binarios definen una relación de grado dos, por ejemplo:

“9 es múltiplo de 3” que lo simbolizamos $P_1^2(9,3)$
“7 es menor que 9” que lo simbolizamos $P_2^2(7,9)$
“Luciana cursa la materia ftc” que lo simbolizamos $P_3^2(\text{Luciana}, \text{ftc})$

- . Ejemplos:

En los siguientes ejemplos el sujeto está subrayado y el resto es predicado.

- a) Sócrates es un hombre.
- b) Juan escribe libros.

Resulta conveniente representar a los predicados con letras mayúsculas y a los sujetos con minúsculas, con lo cual los enunciados de arriba se simbolizan del modo siguiente:

- a) $H(s)$ simboliza «Sócrates es un hombre». También podemos representar que Napoleón es un hombre diciendo $H(n)$, donde n simboliza a Napoleón.
- b) $E(j)$ simboliza « Juan escribe libros», donde j simboliza a Juan, mientras que E simboliza a la propiedad de «escribir libros». También podemos simbolizar que Sócrates escribe libros diciendo $E(s)$.

También podemos ponerle nombres más genéricos a los predicados, por ej:

- a) $P_1^1(x)$ simboliza « x es un hombre»
- b) $P_2^1(x)$ simboliza « x escribe libros»

Representación del Conocimiento:

PREMISAS.

- a) Luciana cursa ftc.
- b) ftc es una materia de Sistemas.

CONCLUSIÓN.

Luciana es alumna de Sistemas.

CUANTIFICADORES

- . Se necesita algo más que un análisis de sujeto y predicado, ya que el significado del enunciado depende de la fuerza, por ejemplo de la palabra “todos”. Se nota $(\forall x)$.
- . También es necesario por representar la frase corriente “Existe al menos un...”. Se nota $(\exists x)$.

Relación entre \forall y \exists :

Ejemplo:

«No todas las aves vuelan» $\sim(\forall x)(A(x) \rightarrow V(x))$

«Algunas aves no vuelan» $(\exists x)(A(x) \wedge \sim V(x))$

Estos ejemplos ilustran un esquema común (pero que puede no cumplirse):

El cuantificador universal va seguido de una implicación, debido a que los enunciados universales suelen ser de la forma, «*dado un x cualquiera, si tiene la propiedad A entonces tiene también la propiedad B*»

El cuantificador existencial va seguido de una conjunción, debido a que los enunciados existenciales suelen ser de la forma, «*existe al menos un x, que tiene la propiedad A y tiene también la propiedad B*»

SINTAXIS

- . Es el lenguaje simbólico de la lógica.
- . Para estudiar los principios del razonamiento, la lógica necesita en primer término capturar y formalizar las estructuras del lenguaje natural en un lenguaje simbólico, para luego formalizar los mecanismos de razonamiento que se aplican sobre dichas estructuras lingüísticas.
- . El lenguaje simbólico consta de:
 - **El alfabeto o vocabulario** → Es el conjunto de símbolos primitivos que pertenecen al lenguaje.

El alfabeto del lenguaje está formado por:

- ✓ Un conjunto de símbolos de constantes $C = \{c_1, c_2, \dots\}$.
- ✓ Un conjunto de símbolos de variables $X = \{x_1, x_2, \dots\}$.
- ✓ Un conjunto de símbolos de funciones $F = \{f_1^1, f_2^1, \dots, f_1^2, f_2^2, \dots\}$.
- ✓ Un conjunto de símbolos de predicados $P = \{P_1^1, P_2^1, \dots, P_1^2, P_2^2, \dots\}$.
- ✓ Símbolos de conectivas (los mismos de la lógica proposicional): $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.
- ✓ Paréntesis de apertura y cierre.
- ✓ El cuantificador universal \forall (“para todo”) y el cuantificador existencial \exists (“existe”).

- **La gramática** → Consiste en un conjunto de reglas que definen recursivamente las cadenas de símbolos que pertenecen al lenguaje.

La gramática del lenguaje define dos clases de elementos, por un lado los **términos** los cuales definen cosas o sujetos, que son las expresiones que denotan los objetos del dominio, y por el otro las **fórmulas bien formadas (fbf)** las cuales definen los predicados por lo que tomarán un valor V o F, con las que se expresan las relaciones entre los objetos.

Los términos se definen inductivamente de la siguiente manera:

- ✓ Los símbolos de constantes y de variables son términos.
- ✓ Si t_1, \dots, t_n son términos y f_i^n es un símbolo de función, entonces $f_i^n(t_1, \dots, t_n)$ es un término.
- ✓ Sólo las expresiones que pueden ser generadas mediante las cláusulas i y ii en un número finito de pasos son términos.

Por su parte, las fórmulas bien formadas se definen así:

- ✓ Si t_1, \dots, t_n son términos y P_i^n es un símbolo de predicado, entonces $P_i^n(t_1, \dots, t_n)$ es una fórmula bien formada. En este caso se denomina fórmula atómica o directamente átomo.
- ✓ Si A y B son fórmulas bien formadas, entonces $(\neg A), (A \wedge B), (A \vee B), (A \rightarrow B)$ y $(A \leftrightarrow B)$ también lo son.
- ✓ Si A es una fórmula bien formada y x es un símbolo de variable, entonces $(\forall x) A$ y $(\exists x) A$ son fórmulas bien formadas.
- ✓ Sólo las expresiones que pueden ser generadas mediante las cláusulas i a iii en un número finito de pasos son fórmulas bien formadas.

Ejemplo:

Los Símbolos del alfabeto del lenguaje de los números:

- c_1 será el símbolo de constante para representar el cero. x será un símbolo de variable.
 f_1^1 será el símbolo de función para representar el sucesor. f_1^2 será el símbolo de función para representar la suma.
 P_1^1 será el símbolo de predicado para representar la propiedad de ser par.
 P_1^2 será el símbolo de predicado para representar la relación de igualdad.
 P_2^2 será el símbolo de predicado para representar la relación <.

Ejemplos de Términos:

- ✓ Los símbolos de constantes y de variables son términos.
 - Entonces c_1 y x son términos.
 - Si t_1, \dots, t_n son términos y f_i^n es un símbolo de función, entonces $f_i^n(t_1, \dots, t_n)$ es un término.
 - Como c_1 es un término, y f_1^1 es un símbolo de función unario, entonces $f_1^1(c_1)$ es un término.
Que podría interpretarse como la función sucesor aplicada al cero, $\text{succ}(0)$.
 - Como c_1 y x son términos, y f_1^2 es un símbolo de función binario, entonces $f_1^2(c_1, x)$ es un término
Que podría interpretarse como la función suma, $+ (0, x)$.
Notar: notación prefija $+ (0, x)$ vs. notación infixia $(0 + x)$.

Ejemplos de fórmulas bien formadas (fbf):

- ✓ Si t_1, \dots, t_n son términos y P_i^n es un símbolo de predicado, entonces $P_i^n(t_1, \dots, t_n)$ es una fbf (en este caso es atómica).
 - Como c_1 es un término y P_1^1 es un símbolo de predicado, entonces $P_1^1(c_1)$ es una fbf.
Que podría interpretarse como la afirmación «el número cero es par».
 - Como c_1 y x son términos y P_1^2 es un símbolo de predicado, entonces $P_1^2(x, c_1)$ es una fbf.
Que podría interpretarse como la afirmación «x es igual a 0».
 - Como c_1 y $f_1^2(c_1, x)$ son términos y P_2^2 es un símbolo de predicado, entonces $P_2^2(f_1^2(c_1, x), x)$ es una fbf.
Que podría interpretarse como la afirmación « $0+x$ es igual a x ».
- Como c_1 y $f_1^1(c_1)$ son términos, entonces $P_1^2(f_1^1(c_1), c_1)$ es una fbf.
Que podría interpretarse como «El sucesor del cero es igual a cero».
- ✓ Si A y B son fórmulas bien formadas, entonces $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$ y $(A \leftrightarrow B)$ también lo son.
 - $\neg P_1^2(f_1^1(c_1), c_1)$ es una fbf.
Que podría interpretarse como «El sucesor del cero no es igual a cero».
- ✓ Si A es una fórmula bien formada y x es un símbolo de variable, entonces $(\forall x) A$ y $(\exists x) A$ son fórmulas bien formadas.
 - $(\forall x) P_1^2(f_1^1(x, c_1), x)$ es una fbf.
Que podría interpretarse como «el cero es el neutro de la suma».
 - $(\forall x) P_2^2(f_1^1(x), x)$ es una fbf.
Que podría interpretarse como « $\text{succ}(x) > x$ ».

. Scope, Ligadura, variable Libre:

Los cuantificadores tienen un *alcance* o *scope*, o *radio de acción* determinado.

Por ejemplo, en la fórmula bien formada:

$$(\forall x) (P_1^1(x) \rightarrow P_2^1(x))$$

Las dos ocurrencias del símbolo de variable x suceden dentro del alcance del cuantificador Mientras que en la fórmula bien formada:

$$(\forall x) P_1^1(x) \rightarrow P_2^1(x)$$

El \forall alcanza sólo a la primera ocurrencia de x. Diremos en el primer caso que x está ligada, y en el segundo que la primera x está *ligada* y la otra está *libre* (por lo que podría sustituirse por cualquier otro símbolo de variable).

Una fbf es *abierta* si contiene algún símbolo de variable libre, y *cerrada* si todos los símbolos de variables están ligados.

Clase 12 | Lógica de Predicados de Primer Orden - Semántica

. Para poder describir los aspectos semánticos de la lógica de predicados, necesitamos antes, introducir el concepto de dominio.

DOMINIO DE INTERPRETACIÓN

. Un dominio está constituido por:

- Un universo U , que es un conjunto no vacío de objetos.
- Un conjunto finito F de funciones, cada una de las cuales asigna a una determinada cantidad de objetos o argumentos de U un objeto de U :

$$F = \{\hat{f}_1^1, \hat{f}_2^1, \dots, \hat{f}_1^2, \hat{f}_2^2, \dots\}$$

El símbolo \hat{f}_j^n denota la j -ésima función de n argumentos. $\hat{f}_i^n : U^n \rightarrow U$

- Un conjunto finito P de relaciones entre los objetos de U :

$$P = \{\hat{P}_1^1, \hat{P}_2^1, \dots, \hat{P}_1^2, \hat{P}_2^2, \dots\}$$

El símbolo \hat{P}_j^n denota la j -ésima relación de grado n . $\hat{P}_i^n \subseteq U^n$

. Ejemplo:

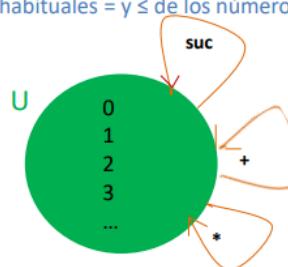
➤ $U = \mathbb{N}$, es decir que U es el conjunto de los números naturales.

➤ $F = \{\text{suc}, +, *\}$ o en general $F = \{f_1^1, f_1^2, f_2^2\}$

F es un conjunto con tres elementos, la función suc , la función suma y multiplicación , con $\text{suc} = \{(x, y) \mid y = x + 1\}$

➤ $P = \{=, \leq\}$ o en general $P = \{P_1^2, P_2^2\}$

P es el conjunto que tiene las relaciones habituales $=$ y \leq de los números naturales, que asumimos definidas.



. Las definiciones de las funciones y relaciones pueden ser *extensionales*, cuando se efectúan enumerando las tuplas que las componen, o *intencionales*, si se establecen a partir de otras funciones o relaciones.

. Ejemplo 2:

➤ $U = \{\text{Alex, Tomás, Catty, Florencia}\} \cup N$. U es un conjunto de niños y números.

➤ $F = \{\text{edad, altura}\}$

siendo:

$\text{edad} = \{(\text{Alex}, 13), (\text{Tomás}, 15), (\text{Catty}, 14), (\text{Flor}, 15)\}$

$\text{altura} = \{(\text{Alex}, 174), (\text{Tomás}, 176), (\text{Catty}, 168), (\text{Flor}, 164)\}$

➤ $P = \{\text{juega-básquet, toca-piano, más-alto, más-joven, amigos}\}$

siendo:

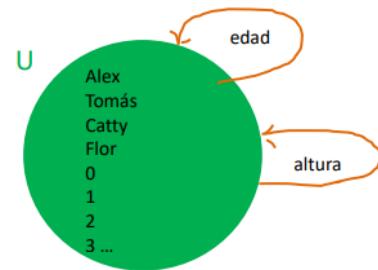
$\text{juega-básquet} = \{\text{Tomás}\}$

$\text{toca-piano} = \{\text{Alex, Catty}\}$

$\text{más-alto} = \{(x, y) \mid \text{altura}(x) > \text{altura}(y)\}$

$\text{más-joven} = \{(x, y) \mid \text{edad}(y) > \text{edad}(x)\}$

$\text{amigos} = \{(\text{Alex, Catty}), (\text{Alex, Flor}), (\text{Catty, Alex}), (\text{Flor, Tomás})\}$



Obsérvese que las nociones de verdad y falsedad, en la lógica de predicados están directamente implícitas en el dominio. Por ejemplo: para establecer la verdad de que Alex toca el piano, incluimos a Alex en el conjunto de niños que tocan el piano, que es la manera extensional de definir dicha propiedad.

INTERPRETACIÓN

. Dados un lenguaje y un dominio, una interpretación I es una función que hace corresponder a los elementos del lenguaje con los elementos del dominio, satisfaciendo las siguientes condiciones:

- Si c_i es un símbolo de constante, entonces $I(c_i) \in U$ (los símbolos de constantes representan objetos del universo del discurso).
- Si f_i^n es un símbolo de función de grado n , entonces $I(f_i^n) = U^n \rightarrow U$ (los símbolos de función representan funciones del dominio).
- Si P_i^n es un símbolo de predicado de grado n , entonces $I(P_i^n) \subseteq U^n$ (los símbolos de predicado representan relaciones del dominio).

. Por lo tanto, una interpretación formaliza la noción de que los símbolos representan las abstracciones de la realidad modelada en el dominio correspondiente.

. Ejemplo:

Dado un lenguaje con los símbolos c_1, f_1^1, f_1^2, f_2^2 y P_1^2

Lo interpretamos en un Dominio con:

Universo: N

$F=\{+, \text{suc}, *\}$

$P=\{=\}$

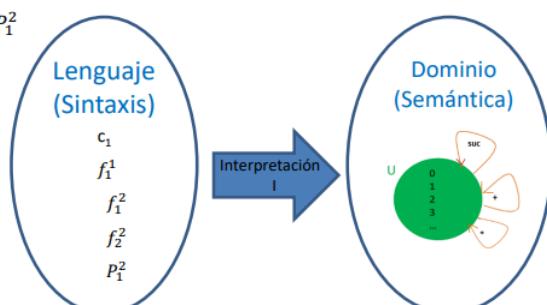
$I(c_1)$ es el cero de los naturales.

$I(f_1^1)$ es la función sucesor en los naturales.

$I(f_1^2)$ es la función suma en los naturales.

$I(f_2^2)$ es la función multiplicación.

$I(P_1^2)$ es la relación de igualdad.



Bajo esta interpretación sobre el dominio de los naturales:

$I(c_1)$ es el cero de los naturales.

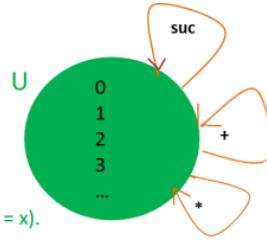
$I(f_1^2)$ es la función suma en los naturales.

$I(P_1^2)$ es la relación de igualdad.

Formulamos algunas propiedades :

$(\forall x) P_1^2(f_1^2(x, c_1), x)$. El cero es el neutro de la suma, es decir: $(\forall x)(x + 0 = x)$.

$(\forall x)(\forall y) P_1^2(f_1^2(x, y), f_1^2(y, x))$. La suma es conmutativa, es decir: $(\forall x)(\forall y)(x + y = y + x)$.



Si bien ésta es la interpretación "estándar", nada, salvo el sentido común, nos impide plantear otras interpretaciones de los símbolos. Por ejemplo, podríamos establecer:

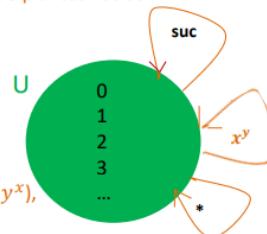
$I'(f_1^2)$ es la función potencia en los naturales, con potencia(x, y) = x^y

Bajo esta nueva interpretación, entonces, quedaría formulado lo siguiente:

$(\forall x) P_1^2(f_1^2(x, c_1), x)$. El cero es el neutro de la potencia, es decir: $(\forall x)(x^0 = x)$,

$(\forall x)(\forall y) P_1^2(f_1^2(x, y), f_1^2(y, x))$. La potencia es conmutativa, es decir: $(\forall x)(\forall y)(x^y = y^x)$,

Ninguna se cumple en el dominio considerado.



Observemos que:

- Los símbolos son sólo símbolos, carentes de «significado» .
- Los símbolos obtienen un significado al ser interpretados sobre un Dominio Semántico.
- El mismo símbolo puede interpretarse de distintas formas (no simultáneamente).

. Queda claro que si nos preguntamos acerca del «valor de verdad» de la siguiente fórmula:

$$(\forall x) P_1^2(f_1^2(x, c_1), x)$$

Nada podemos afirmar sin antes definir un dominio e interpretar el lenguaje.

VALORACIÓN

. Debemos establecer una manera de asignar objetos a todos los términos del lenguaje. Esto se logra por medio de una función denominada valoración (en una interpretación).

. Una valoración v , queda completamente especificada indicando cómo se valoran los símbolos de variables, es decir $v(x_1), v(x_2), \dots$, dado que se define:

- Si c_i es un símbolo de constante, $v(c_i) = I(c_i)$.
- Si f_i^n es un símbolo de función, $v(f_i^n(t_1, \dots, t_n)) = I(f_i^n)(v(t_1), \dots, v(t_n))$.

. Por ejemplo, en la interpretación estándar de los naturales, fijando $v(x) = 7$, la valoración del término $f_1^1(f_1^2(x, c_1))$ se obtiene de la siguiente manera:

$$v(f_1^1(f_1^2(x, c_1))) = I(f_1^1)(v(f_1^2(x, c_1))) = suc(v(f_1^2(x, c_1))) = suc(I(f_1^2)(v(x), v(c_1))) =$$

$$suc(+v(x), v(c_1))) = suc(+7, 0) = suc(7) = 8$$

SATISFACCIÓN

. Para denotar que una fórmula A se satisface con una interpretación I y una valoración v , escribiremos: $I =_{I,v} A$.

. La definición inductiva de la satisfacción de una fórmula es la siguiente:

$$I =_{I,v} P(t_1, \dots, t_n) \text{ si y sólo si } (v(t_1), \dots, v(t_n)) \in I(P)$$

. A partir de la definición de satisfacción para una fbf atómica definimos inductivamente la noción de satisfacción para todas las fbfs del lenguaje.

Sean A y B fbfs del lenguaje:

$\models_{i,v} (\neg A)$	si y sólo si no es el caso que $\models_{i,v} A$
$\models_{i,v} (A \wedge B)$	si y sólo si $\models_{i,v} A$ y $\models_{i,v} B$
$\models_{i,v} (A \rightarrow B)$	si y sólo si no es el caso que $\models_{i,v} A$ y no $\models_{i,v} B$
$\models_{i,v} (\forall x_i) A$	si y sólo si para toda valoración w , i -equivalente, se cumple $\models_{i,w} A$ Es decir que A es verdadera cualquiera sea la valoración de x_i .
$\models_{i,v} (\exists x) A$	si y sólo si para alguna valoración w , i -equivalente, se cumple $\models_{i,w} A$ En este caso alcanza con que A sea verdadera en alguna valoración particular de x .

. Una valoración w es i -equivalente a otra valoración v , cuando $w(x_j)=v(x_j)$, para todo j tal que $j \neq i$.

	x1	x2	x3
v_1	1	2	1
v_2	3	2	1
v_3	2	2	1

$v_1 v_2 v_3$ son 1-equivalentes entre ellas

VERDAD / VALIDEZ

. Si una fórmula es verdadera en toda interpretación es LÓGICAMENTE VÁLIDA.

La notación es $I = A$.

Por ejemplo, $((\forall x) P(x) \rightarrow (\exists x) P(x))$ es válida, cualquiera sea P .

Particularmente, una fórmula válida cuya estructura se corresponde con una tautología de la lógica proposicional, es una tautología de la lógica de predicados. Las fórmulas válidas no proporcionan información alguna de un dominio.

. Si una fórmula A , con una interpretación I , se satisface en todas las valoraciones, se dice que A es VERDADERA en I .

Se escribe así: $I \models A$.

Por ejemplo, $(\forall x) P_1^2(f_1^2(x, c_1), x)$ es verdadera en la interpretación de los naturales con la suma y la igualdad.

Porque todo número natural x cumple que $x + 0 = x$.

. Una fórmula A es SATISFACTIBLE cuando existe una interpretación I y una valoración v , donde $I \models_{i,v} A$.

. Como contrapartida, decimos que una fórmula A es FALSA en una interpretación I si no existe ninguna valoración en I que la satisfaga.

. Las fórmulas falsas en toda interpretación se identifican como CONTRADICCIONES. Por ejemplo, la negación de cualquier fórmula lógicamente válida.

Clase 13 | Verificación Axiomática de Programas

. ¿Cómo probamos un Programa?

- Podemos **Verificarlo**, lo cual es una actividad formal.
- Podemos **Validarlo**, lo cual es una actividad informal y su principal método es el testing.
“El testing asegura la presencia de errores pero no su ausencia”.

. Dijkstra dice: “Pensar cómo sería la prueba de un programa, y luego construirlo siguiendo la estructura de la prueba, para así construirlo y probarlo al mismo tiempo y obtener un programa correcto por construcción”.



VERIFICAR UN PROGRAMA

- . Los programas se pueden verificar por la vía *Semántica* o por la vía *Sintáctica*.
- . Para un programa S_{SWAP} que permuta x con y:

```

 $S_{swap} ::$ 
z := x ;
x := y ;
y := z

```

- Por la vía semántica, usando la semántica de las instrucciones del lenguaje.

variables	programa
1) x = X, y = Y	$z := x ; x := y ; y := z$
2) x = X, y = Y, z = X	$x := y ; y := z$
3) x = Y, y = Y, z = X	$y := z$
4) x = Y, y = X, z = X	

- Por la vía sintáctica, usando axiomas y reglas de un método lógico-deductivo.

- 1) $\{x = X \wedge y = Y\} z := x \{z = X \wedge y = Y\}$ ASI
- 2) $\{z = X \wedge y = Y\} x := y \{z = X \wedge x = Y\}$ ASI
- 3) $\{z = X \wedge x = Y\} y := z \{y = X \wedge x = Y\}$ ASI
- 4) $\{x = X \wedge y = Y\} z := x ; x := y ; y := z \{y = X \wedge x = Y\}$ SEC 1,2,3

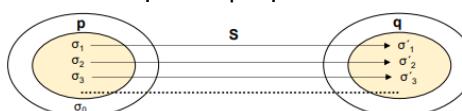
La verificación semántica se complica mucho cuando los programas son complejos (sobre todo concurrentes).

Lo básico que se exige de un método axiomático es que sea *sensato*: que no pruebe enunciados falsos.

Es ideal que también sea *completo*: que pruebe todos los enunciados verdaderos.

. Definiciones:

- $\{x = X \wedge y = Y\} S_{swap} \{y = X \wedge x = Y\}$ es una terna de Hoare o fórmula de correctitud.
- El predicado $x = X \wedge y = Y$ es la precondición de S_{swap} .
- El predicado $y = X \wedge x = Y$ es la postcondición de S_{swap} .
- El par $(x = X \wedge y = Y, y = X \wedge x = Y)$ es la especificación de S_{swap} .
- Un estado σ es una función que asigna a toda variable un valor. Por ejemplo: $\sigma(x) = 1$, $\sigma(y) = 2$, etc.
- Un estado σ satisface un predicado p , si p evaluado con σ es verdadero. Se expresa así: $\sigma \models p$. Por ejemplo: si $\sigma(x) = 1$ y $\sigma(y) = 2$, entonces $\sigma \models x < y$.
- Un programa S es correcto con respecto a una especificación (p, q) , lo que se expresa con $\{p\} S \{q\}$, si para todo estado σ , si $\sigma \models p$ entonces S ejecutado a partir de σ termina en un estado σ' tal que $\sigma' \models q$.

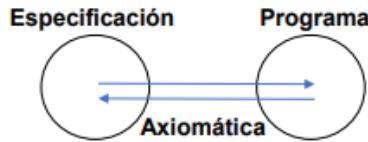


P. ej., si $p = (x = X > 0) \wedge q = (y = 2 \cdot X)$, S debe hacer:
Si $\sigma_1 \models x = 1$, entonces $\sigma'_1 \models y = 2$.
Si $\sigma_2 \models x = 2$, entonces $\sigma'_2 \models y = 4$.
Si $\sigma_3 \models x = 3$, entonces $\sigma'_3 \models y = 6$.
Etc.

¿Se cumple $\{p\} S \{q\}$ si desde σ_0 , S no alcanza un σ' dentro de q ? Sí, porque σ_0 está fuera del alcance de p .

COMPONENTES DEL METODO DE VERIFICACION AXIOMÁTICA

. 1. Lenguaje de Programación (programas con while):



- Instrucciones: $S ::= x := e \mid S_1; S_2 \mid \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi} \mid \text{while } B \text{ do } S_1 \text{ od}$
- Expresiones de tipo entero: $e ::= n \mid x \mid (e_1 + e_2) \mid (e_1 - e_2) \mid (e_1 \cdot e_2) \mid \dots$ n es una constante entera. x es una variable entera.
- Expresiones de tipo booleano: $B ::= \text{true} \mid \text{false} \mid (e_1 = e_2) \mid (e_1 < e_2) \mid \dots \mid \neg B_1 \mid (B_1 \vee B_2) \mid (B_1 \wedge B_2) \mid \dots$

. 2. Lenguaje de Especificación (lógica de predicados):

$p ::= \text{true} \mid \text{false} \mid (e_1 = e_2) \mid (e_1 < e_2) \mid \dots \mid \neg p \mid (p_1 \ p_2) \mid \dots \mid x: p \mid x: p$

. 3. Axiomática:

1. Axioma la Asignación (ASI)

$\{p(e)\} x := e \{p(x)\}$

Si luego de $x := e$ vale p para x, entonces antes de $x := e$ valía p para e.

Por ejemplo: $\{y > 0\} x := y \{x > 0\}$

El axioma de la asignación (ASI): $\{p[x|e]\} x := e \{p\}$ se lee de atrás para adelante. Es más natural una forma hacia adelante: $\{\text{true}\} x := e \{x = e\}$, donde true representa cualquier estado. Pero esta fórmula no siempre es verdadera: $\{\text{true}\} x := x + 1 \{x = x + 1\}$ es una fórmula falsa.

2. Regla de la Secuencia (SEC)

$$\frac{\{p\} S_1 \{r\}, \{r\} S_2 \{q\}}{\{p\} S_1 ; S_2 \{q\}}$$

$$\{p\} S_1 ; S_2 \{q\}$$

El predicado r actúa como nexo y luego se descarta, no se propaga.

3. Regla del condicional (COND)

$$\frac{\{p \wedge B\} S_1 \{q\}, \{p \wedge \neg B\} S_2 \{q\}}{\{p\} \text{ if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{q\}}$$

$$\{p\} \text{ if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{q\}$$

La regla del condicional (COND) formula un modo de verificar una selección condicional fijando un único punto de entrada y un único punto de salida, correspondientes a p y q, respectivamente.

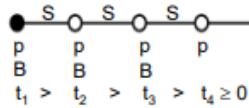
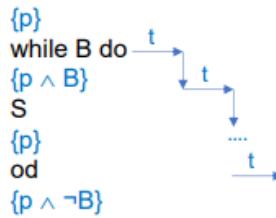
4. Regla de la Repetición (REP)

$$\frac{\{p \wedge B\} S \{p\}, \{p \wedge B \wedge t = Z\} S \{t < Z\}, p \rightarrow t \geq 0}{\{p\} \text{ while } B \text{ do } S \text{ od } \{p \wedge \neg B\}}$$

$$\{p\} \text{ while } B \text{ do } S \text{ od } \{p \wedge \neg B\}$$

p vale antes y después de toda iteración (**invariante**).

t decrece después de toda iteración (**variante**).



5. Regla de la Consecuencia (CONS)

$$r \rightarrow p, \{p\} S \{q\}, q \rightarrow s$$

$$\hline \{r\} S \{s\}$$

La regla de consecuencia (CONS) permite reforzar precondiciones y debilitar postcondiciones. Por ejemplo: de $\{x > 0\} S \{x = 0\}$ y $(x > 5 \rightarrow x > 0)$ se deduce: $\{x > 5\} S \{x = 0\}$

COMPOSICIONALIDAD

- El método de prueba presentado es composicional: Dado un programa S , compuesto por subprogramas S_1, \dots, S_n , que valga la fórmula $\{p\} S \{q\}$ depende sólo de que valgan fórmulas $\{p_i\} S_i \{q_i\}, \dots, \{p_n\} S_n \{q_n\}$, sin importar el contenido de los S_i (noción de caja negra).
- Por ejemplo, dado el programa $S :: S_1; S_2$, si se cumplen las fórmulas: $\{p\} S_1 \{r\}$ y $\{r\} S_2 \{q\}$, también se cumple la fórmula: $\{p\} S_1; S_2 \{q\}$, independientemente del contenido de S_1 y S_2 .
- Más aún, si en lugar de S_2 utilizamos un subprograma S_3 que también satisface la fórmula: $\{r\} S_3 \{q\}$, entonces también se cumple la fórmula: $\{p\} S_1; S_3 \{q\}$, lo que significa que S_2 y S_3 son intercambiables (son funcionalmente equivalentes respecto de (r, q)).



- En los programas concurrentes la composicionalidad se pierde.

ESPECIFICACIONES

- Hemos especificado un programa para calcular el factorial de la siguiente forma:

$$(x > 0, y = x!)$$

Pero la especificación no es correcta. Por ejemplo, el programa $S :: x := 1; y := 1$ satisface ($x > 0, y = x!$) pero no es el programa pedido:

$$\begin{aligned} &\{x = 5\} \\ &x := 1; y := 1 \quad (\text{el resultado tiene que ser } y = 5! = 120) \\ &\{y = 1\} \end{aligned}$$

- Lo que sucede es que las variables de la precondición pueden modificarse a lo largo del programa.
- Lo que se hace es utilizar variables lógicas, para congelar valores. En el ejemplo considerado haríamos: $(x = X \wedge X > 0, y = X!)$

Lo que no se puede hacer es agregar a la especificación que la variable x no se modifique nunca, ya que la lógica de predicados no lo permite. Una lógica que sí lo permite es la temporal.

SENSATEZ, COMPLETITUD Y AUTOMATIZACIÓN DE LA VERIFICACIÓN

Sensatez

- Si se prueba $\{p\} S \{q\}$,
se debe cumplir $\{p\} S \{q\}$.

Propiedad **obligatoria**.

Completitud

- Si se cumple $\{p\} S \{q\}$,
se debe poder probar $\{p\} S \{q\}$.

Propiedad **deseable**.

- La completitud depende de la expresividad del lenguaje de especificación. Por ejemplo:
Dada la fórmula $\{p\} S_1 ; S_2 \{q\}$, debe poder encontrarse un predicado, tal que $\{p\} S_1 \{r\} ; S_2 \{q\}$.

Automatización

- La verificación de programas es en general **indecidible**, y por lo tanto **no automatizable**.
- Existen numerosas **herramientas** que asisten interactivamente al programador.
- Para los programas de estados finitos hay sistemas automáticos (**model checking**) con lógica temporal.