Resumen Computabilidad y Complejidad Computacional

Definiciones Generales

Conceptos matemáticos

Alfabeto: Conjunto finito de símbolos, ejemplo: $\sum = \{a, b, c\}$.

Lenguaje: Conjunto de cadenas formadas a partir de un *alfabeto*, ejemplo, un lenguaje L con alfabeto Σ es $L = \{aaa, b, ababab, ccb\}$.

- **Σ*** es el lenguaje o conjunto de cadenas de símbolos generado a partir de Σ, es *infinito* pero *sus cadenas son finitas*, incluye a la *cadena vacía* λ.

Generalidades de las Máquinas de Turing

Una **Máquina de Turing** es una *modelización muy simple de una computadora*, aceptada universalmente para estudiar la computabilidad y la complejidad computacional de los problemas.

Problemas Computables Decidibles

Problemas que cuentan con MT que los resuelven **totalmente** (casos positivos y negativos).

Problemas de Búsqueda: Consisten en encontrar una solución a una instancia dada del problema a resolver, la máquina puede devolver una *solución al problema* o en el caso que no exista una solución devolver *"no"*.

Problemas de Decisión (Los que tratamos): Consisten en determinar la existencia de una solución a un problema, siendo las salidas posibles *"si"* o *"no"*.

- Dentro de esta clase de podemos referirnos indistintamente a problemas y lenguajes porque existe una *correspondencia directa y natural entre un problema de decisión y el lenguaje que representa sus instancias positivas*.

Problemas Computables no Decidibles

Son problemas donde las **MT** tiene casos donde no puede decir ni **"si"** ni **"no"** y por lo tanto, **loopea**.

Problemas no Computables

Problemas que **ni siquiera cuentan con MT que responda sí en todos los casos positivos**.

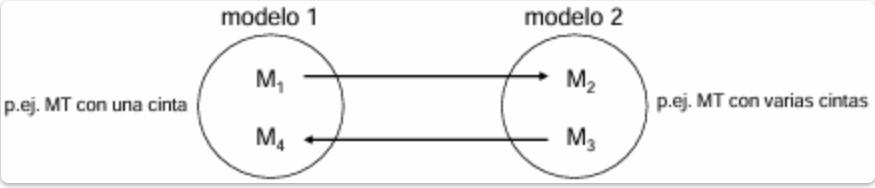
Tesis de Church-Turing

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

Equivalencia de 2 MTs

Dos MT son equivalentes si aceptan el mismo lenguaje.

Dos modelos de MT son equivalentes si dada una MT de un modelo existe una MT equivalente del otro.



Distintas visiones de MTs

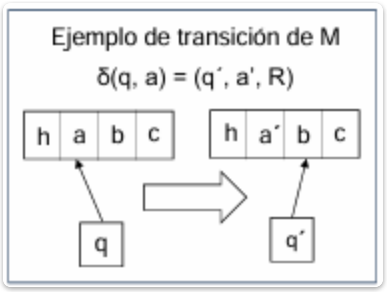
MT calculadora: Es la MT que se usa para los *problemas de búsqueda*.

MT generadora: MT que genera en una cinta de salida *todas las cadenas del lenguaje que acepta*. Por teorema, **existe una MT M que acepta un lenguaje L sii existe una MT M' que genera el lenguaje L.**

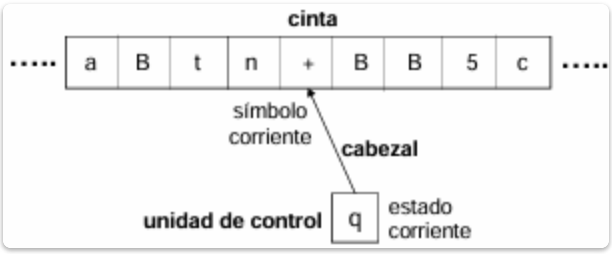
Componentes y Funcionamiento de una MT

Formalmente, definimos a una MT como una **tupla** $(Q, \Gamma, \delta, q_0, q_A, q_R)$

- Q es el conjunto de *estados* de la máquina.
- Γ es el *alfabeto de la máquina*, incluye al blanco (B), este último no siendo admitido en una cadena de entrada.
- q_0 es el *estado inicial* de la máquina.
- q_A y q_R son los *estados finales* de la máquina
- δ es la *función de transición* de la máquina

- 

Componentes de una MT



Las MT pueden tener *varias cintas*, pero se demuestra que una MT con varias cintas **no tiene más potencia computacional** que una MT con 1 cinta.

Para toda MT M_1 con varias cintas, existe una MT M_2 equivalente (acepta el mismo lenguaje) con una cinta. Se prueba que h pasos de M_1 se pueden simular con unos h_2 pasos de M_2 (**retardo cuadrático**).

Qué hacer si nos piden construir una MT por completo

1. Dar la **idea general** de qué va a hacer nuestra MT.
2. Definir el conjunto de **estados** Q de nuestra MT, incluyendo $q_0, q_A \ y \ q_R$. Para los estados adicionales en lo posible dejar claro que hacen.
3. Definir el **alfabeto** Γ incluyendo al Blanco.
4. Definir la **función de transición** δ de la MT. Esto puede ser lo más difícil pero también es mejor dejar en claro que hace cada transición. Podemos definir la *lista* (omitiendo los *casos de rechazo*) o usar la *matriz/tabla* (dejando vacía las celdas que correspondan a los *casos de rechazo*).
5. **OPCIONAL:** Podemos hacer el dibujo de como actuaría la MT para un ejemplo de una cadena w que queramos.

Ejercicio 3

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

- **Idea General:**
  - Recorrer la cadena y determinar si después de "h" hay una "a" o una "b".
    - Si hay una "a" luego de "h" buscamos solo "a" recorriendo hacia la derecha, si encontramos una "b" frenamos y retornamos "no".
    - Si hay una "b" luego de "h" buscamos solo "b" recorriendo hacia la derecha, si encontramos una "a" frenamos y retornamos "no".
    - Si recorremos toda la cadena retornamos "si".

- **Estados:** Q = $\{q_0, q_1, q_a, q_b, q_A, q_R\}$

1. q_0 : Estado inicial.
2. q_1 : M determina que letra hay.
3. q_a : M busca una "a".
4. q_b : M busca una "b".

- **Alfabeto:** Γ = {h, a, b, B}

- **Función de Transición:** δ

1. $\delta(q_0, h) = \{(q_1, h, R)\}$ → Pasamos a determinar que letra hay después.
2. $\delta(q_1, a) = \{(q_a, a, R)\}$ → Encontramos "a" y buscamos más "a".
3. $\delta(q_1, b) = \{(q_b, b, R)\}$ → Encontramos "b" y buscamos más "b".
4. $\delta(q_1, B) = \{(q_A, B, S)\}$ → Encontramos B y aceptamos.
5. $\delta(q_a, a) = \{(q_a, a, R)\}$ → Recorremos siguiendo con las "a".
6. $\delta(q_a, B) = \{(q_A, B, S)\}$ → Llegamos al final y solo había "a".
7. $\delta(q_b, b) = \{(q_b, b, R)\}$ → Recorremos siguiendo con las "b".
8. $\delta(q_b, B) = \{(q_A, B, S)\}$ → Llegamos al final y solo había "b".

- **MT M** = $(Q, \Gamma, \delta, q_0, q_A, q_R)$

Tipos de Lenguajes

Lenguajes Recursivos

Son los lenguajes aceptados por MT que siempre paran ya sea caso negativo o positivo. El conjunto R es el conjunto de estos lenguajes.

Lenguajes Recursivamente Enumerables

Son los lenguajes aceptados por MT que a partir de algunas instancias negativas no paran. El conjunto RE es el conjunto de estos lenguajes.

Lenguajes No Recursivamente Enumerables

Son los lenguajes que ni siquiera llegan a tener una MT que los acepten, es decir, que para a partir de alguna instancia positiva del problema. El conjunto $\mathcal{L}$ es el conjunto de estos lenguajes.

Conjuntos

Se cumple por definición que $R \subseteq RE \subseteq \mathcal{L}$.

- Si un lenguaje pertenece a R tiene una MT que se detiene para las instancias positivas y negativas, es decir, está también incluido en RE porque cumple la definición de detenerse para las instancias positivas, y a su vez, dado que todo lenguaje recursivamente enumerable (pertenece a RE) es un conjunto de cadenas sobre algún alfabeto (el lenguaje aceptado por una MT), todo lenguaje recursivamente enumerable es también un lenguaje, y por lo tanto pertenece al conjunto de todos los lenguajes $\mathcal{L}$.
  Por Diagonalización podemos ser más precisos y demostrar que $R \subset RE \subset \mathcal{L}$.

Conjunto $\mathcal{L}$

Conjunto universal de todos los lenguajes con alfabeto Σ, es el conjunto de todos los subconjuntos de Σ*.

Conjunto R

Conjunto de los lenguajes recursivos. Este es el conjunto en el que nos centramos luego para la Complejidad Computacional.

Propiedades de R

1. Si $L \in R$, entonces $L^C \in R$
   - Esto se debe a que podemos hacer una MT M_2 que use la MT M_1 que decide L pero lo que hace M_2 es invertir los estados finales de M_1, si M_1 dice "si", M_2 dice "no", y viceversa. De esta forma M_2 decide L^C y, por lo tanto, $L^C \in R$.
2. Si $L_1 \in R$ y $L_2 \in R$, entonces $L_1 \cap L_2 \in R$
   - Esto se puede probar construyendo una MT M que ejecute secuencialmente M_1 y M_2 y acepte sii M_1 y M_2 aceptan.
3. Si $L_1 \in R$ y $L_2 \in R$, entonces $L_1 \cup L_2 \in R$
   - Esto se puede probar construyendo una MT M que ejecute secuencialmente M_1 y M_2 y acepte si al menos una de las 2 MT acepta.
4. $R \subseteq RE \cap CO - RE$, y por lo tanto $R = RE \cap CO - RE$
   - $R \subseteq RE$ se ve por definición.
   - $R \subseteq CO - RE$ porque si $L \in R$ entonces $L^C \in R$, por lo que $L^C \in RE$, y así por definición $L \in CO - RE$.

Pertenencia a R

Para probar que un lenguaje L pertenece a R tenemos que **construir** una máquina de Turing que lo **decida** (*pare para las instancias positivas y negativas del problema*).

Los lenguajes **Σ***, $\emptyset$ y todo **lenguaje finito** pertenecen a R.

No Pertenencia a R

Podemos probar la no pertenencia de un lenguaje L a R de dos formas:

- Con *Diagonalización* (muy complicado pero es la base que nos permite hacer la segunda forma).
- Con *Reducciones*, si demostramos que a partir de una lenguaje L_i que sabemos que no está en R existe una reducción hacia L, el lenguaje que sospechamos que no pertenecería a R, entonces L no pertenecería a R.

Conjunto RE

Conjunto de los **lenguajes recursivamente enumerables**.

Propiedades de RE

1. **Si $L_1 \in RE$ y $L_2 \in RE$, entonces $L_1 \cup L_2 \in RE$**
   - Esto se prueba construyendo una MT M que ejecuta *"en paralelo"* M_1 y M_2 (alternando una paso de cada una), y aceptar w sii una de las dos MT acepta w.
2. **Si $L_1 \in RE$ y $L_2 \in RE$, entonces $L_1 \cap L_2 \in RE$**
   - Esto se puede probar construyendo una MT M que ejecute secuencialmente M_1 y M_2 y acepte sii M_1 y M_2 aceptan. Si alguna de las 2 loopea, entonces M también lo hace.
3. **$L \in RE$ sii $L^C \in CO - RE$**
   - Esto se debe a si hacemos una MT M_2 que use la MT M_1 que acepta L de modo que lo que hace M_2 es invertir los estados finales de M_1, si M_1 dice "si", M_2 dice "no", y viceversa, nos puede pasar que hay casos en los que M_1 loopea para entradas que no están dentro de L pero M_2 debería aceptarlas ya que no pertenecen a L pero como M_1 loopea, M_2 hace lo mismo, por lo tanto $L^C \notin RE$.

Pertenencia a RE

Para probar que un lenguaje L pertenece a RE tenemos que **construir** una máquina de Turing que lo **acepte** (pare siempre para las instancias positivas).

No Pertenencia a RE

Podemos probar la no pertenencia de un lenguaje L a RE de dos formas:

- Con *Diagonalización* (muy complicado pero es la base que nos permite hacer la segunda forma).
- Con *Reducciones*, si demostramos que a partir de una lenguaje L_i que sabemos que no está en RE existe una reducción hacia L, el lenguaje que sospechamos que no pertenecería a RE, entonces L no pertenecería a RE.

Conjunto $CO - RE$

Conjunto de los **complementos** de los lenguajes del conjunto RE.

Propiedades de $CO - RE$

1. $L \in CO - RE$ **sii** $L^C \in RE$
2. $RE \cap CO - RE \subseteq R$**, y por lo tanto se cumple la igualdad** $R = RE \cap CO - RE$
   - Esto se prueba viendo que si un problema y el problema contrario son computables, entonces ambos son decidibles.

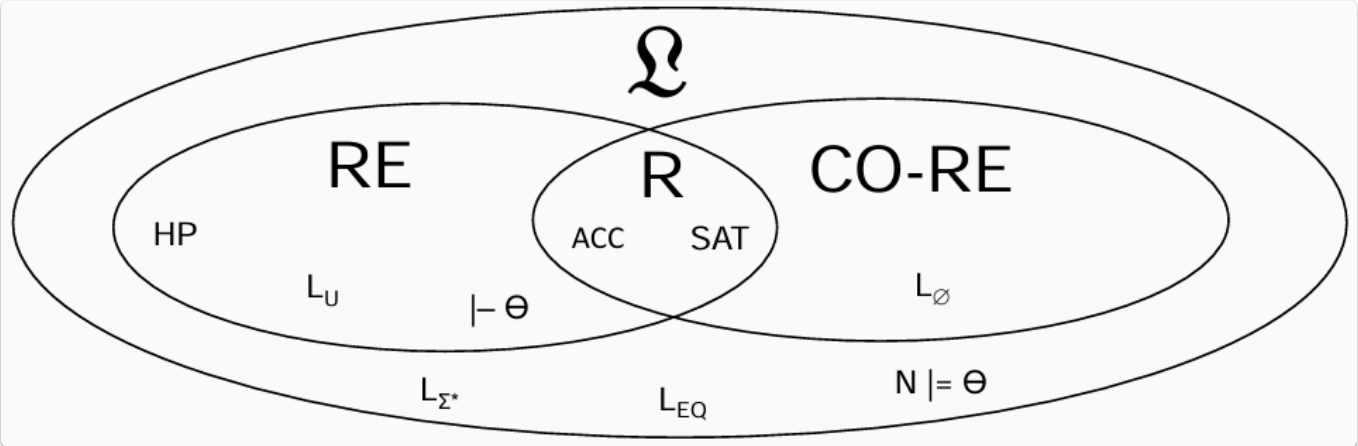
Pertenencia a $CO - RE$

Para probar que un lenguaje L pertenece a $CO - RE$ tenemos que probar que L^C **pertenece a** RE**.**

No Pertenencia a $CO - RE$

Para probar que un lenguaje L no pertenecería a $CO - RE$ tenemos que probar que L^C **no pertenecería a** RE**.**

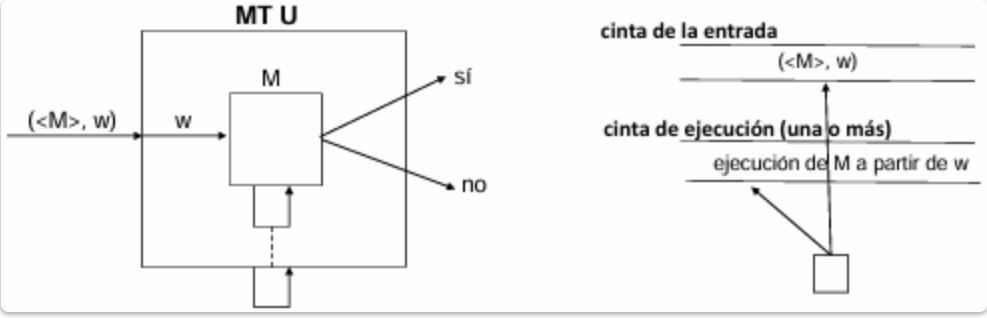
Esquema más completo de la Computabilidad



Máquina de Turing Universal

Una máquina de Turing universal (MT U) es una máquina de Turing **capaz de ejecutar cualquier otra**. La MT U recibe como entrada una **MT** M (codificada mediante una cadena) y una **cadena**

w, y ejecuta M a partir de w.



Los lenguajes más ICONICS

- $HP = \{(< M >, w) \mid \text{M para a partir de w}\}$ está en RE-completo
- $HP^C = \{(< M >, w) \mid \text{M no para a partir de w}\}$ está en CO-RE
- $D = \{w_i \mid M_i \text{ acepta } w_i\}$ considerando el orden canónico, está en RE
- $D^C = \{w_i \mid M_i \text{ rechaza } w_i\}$ considerando el orden canónico, está en CO-RE
- $D_{HP} = \{w_i \mid M_i \text{ para a partir de } w_i\}$ considerando el orden canónico, está en RE
- $D_{HP}^C = \{w_i \mid M_i \text{ no para a partir de } w_i\}$ considerando el orden canónico, está en CO-RE
- $L_U = \{(< M >, w) \mid \text{M acepta w}\}$ está en RE
- $L_U^C = \{(< M >, w) \mid \text{M rechaza w}\}$ está en CO-RE
- $L_{\Sigma*} = \{< M > \mid L(M) = \Sigma*\}$ está en $\mathcal{L}$.
- $L_{\Sigma*}^C = \{< M > \mid L(M) \neq \Sigma*\}$ está en $\mathcal{L}$.
- $L_\emptyset = \{< M > \mid L(M) = \emptyset\}$ está en CO-RE
- $L_\emptyset^C = \{< M > \mid L(M) \neq \emptyset\}$ está en RE
- $SAT = \{\varphi \mid \varphi \text{ es una fórmula booleana satisfactible con m variables en la FNC}\}$ es NP-completo y está en R
- $SAT^C = \{\varphi \mid \varphi \text{ no es una fórmula booleana satisfactible con m variables en la FNC}\}$ es CO-NP-completo y está en R
- $CH = \{G \mid \text{G es un grafo y tiene un circuito de Hamilton}\}$ es NP-completo y está en R
- $CH^C = \{G \mid \text{G es un grafo y no tiene un circuito de Hamilton}\}$ es CO-NP-completo y está en R
- $ISO = \{(G_1, G_2) \mid G_1 \text{ y } G_2 \text{ son grafos isomorfos}\}$ está en NP
- $ISO^C = \{(G_1, G_2) \mid G_1 \text{ y } G_2 \text{ no son grafos isomorfos}\}$ está en CO-NP

Cómo saber si una máquina loopea o no

Esta estrategia se usa originalmente para burlar el Halting Problem, nosotros podemos saber si una máquina de Turing loopea o no si es que podemos hacer que trabaje en un espacio acotado de celdas, de este modo podemos calcular la cantidad máxima de configuraciones

distintas que una MT puede tener, si la máquina loopea, en algún momento repite una configuración.

- Podemos ver el concepto de *configuración* como una *"foto" de un momento específico de la MT* donde capturamos *la posición del cabezal, el estado de la máquina y el contenido de la cinta.*

La fórmula para calcular la cantidad máxima de configuraciones distintas es:
$Total = \text{cantidad de estados posibles} \cdot \text{número de celdas ocupadas por la entrada} + 2 \text{ Blancos} \cdot \text{cantidad de símbolos del alfabeto elevado a la cantidad de celdas ocupadas.}$

Reducciones de Lenguajes

Formalmente:

- Dados dos lenguajes L_1 y L_2, una reducción de L_1 a L_2 es una función total computable $f : \Sigma* \longrightarrow \Sigma*$ (función definida para todas las cadenas de $\Sigma*$ y computable por una MT M_f), que para toda cadena w:
  - $w \in L_1$, entonces $f(w) \in L_2$.
  - $w \notin L_1$, entonces $f(w) \notin L_2$.
  En criollo
- Dados dos lenguajes, existe una reducción de uno al otro si creamos una función f cuya MT M_f cumpla 2 condiciones:
  - Sea *total computable*, es decir, M_f tiene que parar siempre y generar un resultado para toda cadena del alfabeto.
  - tenga *correctitud*, es decir, "de adentro a adentro, de afuera a afuera".

Es la equivalencia a la invocación de una subrutina.

El lenguaje de la derecha es *igual o más complejo* que el de la izquierda, o lo mismo, el lenguaje de la izquierda es *igual o menos complejo* que el de la derecha.

Para la no pertenencia tenemos que ser inteligentes con la elección del lenguaje de la izquierda en la reducción.

En la función de reducción podemos simular MTs que sepamos que paren siempre, si no podemos afirmar que las MTs paran siempre no podemos ejecutarlas, normalmente para estos casos la función de reducción actúa como una traductora sintáctica que no simula nada.

Propiedades de las Reducciones

1. 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 $L_1 \notin R \rightarrow L_2 \notin R$.

2. **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 $L_1 \notin RE \rightarrow L_2 \notin RE$**.
3. **Reflexividad:** Para todo lenguaje L se cumple $L \leq L$.
4. **Transitividad:** Si $L_1 \leq L_2$ y $L_2 \leq L_3$, entonces $L_1 \leq L_3$. Es una *composición de funciones*.
5. $L_1 \leq L_2$ **sii** $L_1^C \leq L_2^C$.
6. **Simetría:** $L_1 \leq L_2$ no implica $L_2 \leq L_1$. Esta propiedad solamente se puede cumplir en ciertos casos donde *ambos lenguajes pertenezcan al conjunto R.*

Qué hacer si tenemos que plantear una reducción

1. Dar la **idea general** de qué va a hacer nuestra reducción o de que forma vamos a afrontar el problema.
2. Especificar la **reducción**, es decir, definir la función f formalmente.
3. Demostrar que la **función de reducción** es **total computable y correcta**.

Ejercicio 3

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

Idea General: Analizando los 2 lenguajes tenemos a $D_{HP} = \{w_i \mid M_i$ para a partir de $w_i\}$ y a $HP = \{(< M >, w) \mid M$ **se detiene a partir de** $w\}$. D_{HP} va a recibir una cadena w_i, al seguir el orden canónico, sabemos que esa cadena w_i va a tener una MT M_i que puede parar o no a partir de esa entrada, nosotros debemos pasarle a HP el par $(< M_i >), w_i)$. **Podemos plantear la reducción de esta forma:**

Reducción: Se define $f(w_i) = (< M_i >, w_i)$, tal que M_i es la MT asociada a la cadena w_i por orden canónico.

Computabilidad: Existe una MT M_f, que computa f: genera el par $(< M_i >, w_i)$, donde $< M_i >$ es la $i-$***ésima*** MT dada por el orden canónico. f es *total* ya que está definida para toda entrada de su dominio $(\Sigma*)$ y siempre encontrará la $i-$***ésima*** MT dada por el orden canónico, por lo tanto, no loopea, y es *computable* ya que la búsqueda de la $i-$***ésima*** MT y la creación del par $(< M_i >, w_i)$ se da en un tiempo finito.

Correctitud:

- $w_i \in D_{HP} \rightarrow M_i$ se detiene a partir de w $\rightarrow f(w_i) = (< M_i >, w_i) \in HP$
- $w_i \notin D_{HP} \rightarrow M_i$ no se detiene a partir de w $\rightarrow f(w_i) = (< M_i >, w_i) \notin HP$

Tiempo polinomial y no polinomial

Las métricas de complejidad computacional que usamos son:

- *Tiempo:* Cantidad de pasos ejecutados por una MT.
- *Espacio:* Cantidad de celdas ocupadas por una MT.

Dentro de los problemas decidibles (recursivos, osea, los de R) ahora clasificamos:

- *Problemas intratables:* No tienen un algoritmo eficiente que los resuelva. Tiempo no polinomial.
- *Problemas tratables:* Tienen un algoritmo eficiente que los resuelve. Tiempo polinomial.

Complejidad Temporal

Nosotros medimos el tiempo con **funciones temporales $T(n)$** porque una MT hace más pasos a medida que su entrada es más grande. **Siempre vamos a tomar el peor de los casos**.

Una función $T1(n)$ **es del orden de una función** $T2(n)$, que se anota así: $T1(n) = O(T2(n))$, **sii para todo** $n \geq n_0$ se cumple $T1(n) \leq c \cdot T2(n)$, con $c > 0$. Esto quiere decir que podemos acotar el orden de una función al de otra mayor **si encontramos un valor de n y de c que cumplan esa desigualdad.** Ejemplo:

- Para probar que $n^3 = O(2^n)$ *si tomamos* $c = 1$ y $n_0 = 10$ nos damos cuenta que a partir de ese punto se cumple la *desigualdad* $n^3 \leq c \cdot 2^n$.

Una MT M tarda tiempo $T(n)$, sii a partir de toda entrada w, con $|w| = n$, M hace a lo sumo $T(n)$ pasos.

Un lenguaje $L \in TIME(T(n))$ sii existe una MT M que lo decide en tiempo $O(T(n))$.

Si una MT con K_1 cintas decide un lenguaje en **tiempo polinomial**, digamos $O(n^k)$ para alguna constante k, entonces existe una MT equivalente con K_2 cintas (incluso una sola cinta) que también decide el mismo lenguaje en **tiempo polinomial**. Lo mismo se aplica para una resolución en **tiempo exponencial**.

En la Complejidad Temporal descartamos la **codificación unaria** para las cadenas ya que **no es físicamente realizable para números grandes, genera inconsistencias y tergiversa la verdadera complejidad temporal de los lenguajes.**

Para lo que viene asumimos que:

- $P \neq NP$.
- $NP \neq EXP$.

Tesis fuerte de Church-Turing

Si L es decidible en tiempo $poly(n)$ por un modelo computacional físicamente realizable, también es decidible en tiempo $poly(n)$ por una MT.

Clase P

Un lenguaje pertenece a la clase P si existe una Máquina de Turing determinística que lo decide en tiempo polinomial, es decir, en tiempo $O(n^k)$ para alguna constante k.

Esta clase contiene a los lenguajes tratables.

Nos damos cuenta que un lenguaje no pertenecería a esta clase si pensando una solución eficiente para resolverlo caemos en el hecho de que la forma más eficiente es por la "fuerza bruta" revisando las exponenciales posibles soluciones.

Propiedades de la clase P

1. Se cumple que $P \subseteq NP$
   - Cualquier lenguaje decidible en tiempo polinomial por una MT determinística también puede ser "verificado" en tiempo polinomial (sin necesidad real de un certificado).
2. Se cumple que $P \subset EXP$.
   - Cualquier lenguaje decidible en tiempo polinomial también es decidible en tiempo exponencial.
3. $L \in P$ sii $L^C \in P$
   - Esto se prueba por construcción de una MT M^C que use M e invierta las salidas de M, y dado que el tiempo de ejecución de M es polinomial, el tiempo de ejecución de M^C también será polinomial.

Clase NP

Un lenguaje pertenece a la clase NP si tiene la siguiente propiedad:

- Si una cadena le pertenece, entonces dicha pertenencia se puede verificar en tiempo polinomial, con la ayuda de otra cadena conocida como certificado que debe ser sucinto (Tiene tamaño polinomial respecto del tamaño de la entrada).

Los certificados tienen que ser sucintos (tener tamaño polinomial respecto al tamaño de la entrada) porque sino se tardaría un tiempo mayor al polinomial en solo leer el certificado lo que haría que nunca una validación se de en tiempo polinomial.

Propiedades de la clase NP

1. Se cumple que $NP \subseteq EXP$

- Si $L \in NP$, entonces existe una máquina M´ que decide L en tiempo exponencial, simplemente probando todos los posibles certificados hasta que alguno haga que M acepte.

2. $L \in NP$ sii $L^C \in CO - NP$.
3. Se acepta la conjetura $P \subset NP \cap CO - NP$.

Clase EXP

Un lenguaje pertenece a la clase EXP (o tiempo exponencial) si es decidible en tiempo $O(c^{poly(n)})$, con c constante, donde $poly(n)$ es una función polinomial de la longitud de la entrada n.

Clase $CO - NP$

Es la clase que tiene a los complementos de los lenguajes NP.

Propiedades de la clase $CO - NP$

1. Se acepta la conjetura $NP \neq CO - NP$, es decir, $L \in CO - NP$ sii $L^C \in NP$
2. Se acepta la conjetura $P \subset NP \cap CO - NP$.

Reducciones Polinomiales

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 ($L1 \leq_p L2$).

Propiedades de las Reducciones Polinomiales

1. Transitividad: Si $L_1 \leq_p L_2$ y $L_2 \leq_p L_3$, entonces $L_1 \leq_p L_3$. Es una *composición de funciones polinomiales*.
2. $L_1 \leq_p L_2$ sii $L_1^C \leq_p L_2^C$.
3. Reflexividad: Para todo lenguaje L se cumple $L \leq_p L$.
4. Simetría: $L_1 \leq_p L_2$ no implica $L_2 \leq_p L_1$.
   - La Simetría se cumple para los lenguajes $NP - completos$.

Qué hacer si tenemos que plantear una reducción polinomial

Hacemos lo mismo que para una reducción común pero ahora también tenemos que justificar que la función de reducción justamente tarda tiempo polinomial.

Ejercicio 4

Sean los lenguajes A y B, tales que $A \neq \emptyset$, $A \neq \Sigma*$, y $B \in P$. Probar: $(A \cap B) \leq_P A$. *Ayuda: intentar con una reducción polinomial que, dada una cadena w, lo primero que haga sea chequear si $w \in B$, teniendo en cuenta que existe un elemento e que no está en A.*

Cosas que sabemos:

- Al $A \neq \emptyset$, $A \neq \Sigma*$ nos damos cuenta que existe al menos una cadena e que no está en A.
- Al $B \in P$ sabemos que existe una máquina de Turing M_B que decide B en tiempo polinomial.

Idea general: Primero tomamos la cadena w y verificamos que $w \in B$ utilizando la máquina M_B que sabemos que decide B en tiempo polinomial, si $w \notin B$ transformamos w a esta cadena e que sabemos que no está en A, si $w \in B$ mantenemos la cadena w para verificar que pertenezca a A.

Reducción: vamos a tener la función $f(w) = w$ si $w \in B$ o e si $w \notin B$. La función se computa en **tiempo polinomial** ya que las tareas que realiza pueden ser:

- Mantener la misma cadena w que es *polinomial*.
- Cambiar w por la constante e que tomamos en cuenta, que también es *polinomial*.

Verificación de la correctitud:

1. Si $w \in A \cap B$:
   - Entonces $w \in B \rightarrow f(w) = w$.
   - Como $w \in A \rightarrow f(w) = w \in A$.
   - *Entonces* $f(w) \in A$.
2. Si $w \notin A \cap B$:
   - Si $w \notin B \rightarrow f(w) = e$, pero $e \notin A$ por construcción $\rightarrow f(w) \notin A$.
   - Si $w \in B$ pero $w \notin A \rightarrow f(w) = w \notin A$.
   - *En ambos casos,* $f(w) \notin A$.

Clase $NP - completo$

Un lenguaje L es $NP - completo$ si cumple dos condiciones:

- L pertenece a la clase NP.
- L es $NP - difícil$. Es decir, todo lenguaje de NP se reduce polinomialmente a L.
  En resumen, *un lenguaje $NP - completo$ es un problema que está en NP y es tan difícil como cualquier otro problema en NP.*

Si $P \neq NP$, un lenguaje $NP - completo$ no pertenece a P porque **si un lenguaje $NP - completo$ L perteneciera a P, entonces se demostraría que $P = NP$.**

- Si L perteneciera a $NP - completo$ y a P, por definición todos los lenguajes de NP se reducen polinomialmente a L, pero como L pertenece a P esto haría que todos los lenguajes de NP pertenecieran a P por propiedad de las reducciones, lo que demostraría que $P = NP$.

Esquema para agregar un Lenguaje a la clase $NP - completo$

El esquema nos dice:

- Demostrar que L_1 pertenece a la clase NP.
- Elegir un lenguaje L que ya se sabe que es $NP-completo$ (como SAT o algún otro lenguaje previamente probado como $NP-completo$) y construir una reducción polinomial de L a L_1. Esto implica encontrar una función f computable en tiempo polinomial tal que para toda cadena w, $w \in L$ si y solo si $f(w) \in L_1$.

Propiedades de la clase $NP-completo$

1. Todo par de lenguajes L_1 y L_2 $NP-completos$ conocidos son **p-isomorfos**:
   - Existe una reducción polinomial $L_1 \leq_p L_2$ que usa f y también existe una reducción polinomial $L_2 \leq L_1$ que usa f^{-1} (la función inversa a f). Si todos los lenguajes de esta clase son p-isomorfos, se cumple $P \neq NP$.
2. Los lenguajes de esta clase son conocidos como **densos** (tienen muchas cadenas, formalmente $exp(n)$ con longitud a lo sumo n), en la **complejidad computacional** el **tamaño de un lenguaje está muy ligado a su dificultad** (más grande, más difícil). Se prueba que si existe un lenguaje $NP-completo$ **no denso** (disperso), se cumple $P = NP$.
3. En la mayoría de estos lenguajes, dado dos lenguajes $NP-completos$ conocidos L_1 y L_2 en una reducción polinomial $L_1 \leq_p L_2$ se cumple 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** (son **parsimoniosas**).

Clase $NP-difícil$

Un lenguaje L se considera $NP-difícil$ si todo lenguaje de la clase NP se puede **reducir polinomialmente** a él.

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**.

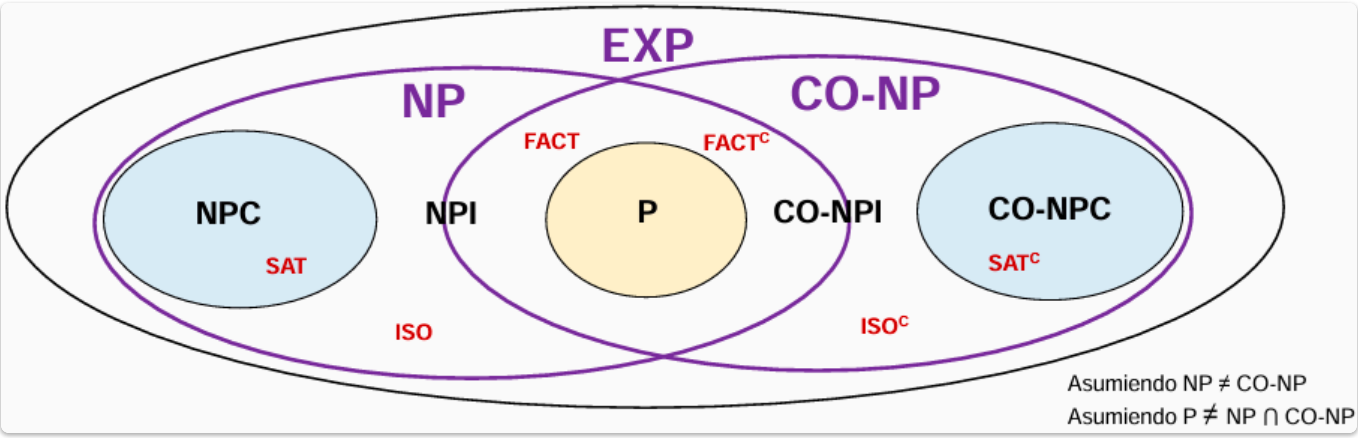
Sospechamos que un lenguaje de NP está en NPI cuando:

- Se cree que el lenguaje *no puede ser decidido por una Máquina de Turing en tiempo polinomial*, lo que sugiere que no pertenecería a la clase P.
- No se ha encontrado una *reducción polinomial desde algún lenguaje conocido que sea $NP-completo$ hacia este lenguaje*, lo que sugiere que no pertenecería a la clase $NP-completo$.

Clase $CO - NP - completo$

Un lenguaje es $CO - NP - completo$ sii todos los lenguajes de $CO - NP$ se reducen polinomialmente a él. Acá están los complementos de los lenguajes pertenecientes a $NP - completo$.

Esquema más completo de la Complejidad Temporal



Complejidad Espacial

Una MT M ocupa espacio $S(n)$ sii 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).

Acá también se cumple la robustez, dada una MT M_1 de espacio $S(n)$ con varias cintas de trabajo, existe una MT M_2 equivalente de espacio $S(n)$ con una sola cinta de trabajo.

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

En la complejidad espacial, el espacio que puede ocupar un contador en binario es de $O(log_2(n))$.

¿Por qué si una MT tarda tiempo $poly(n)$ entonces ocupa espacio $poly(n)$?

Una MT que se ejecuta en tiempo $poly(n)$, es decir, hace $T(n)$ pasos, en el peor caso, solo puede acceder y, por lo tanto, ocupar a lo sumo T(n) celdas en sus cintas de trabajo. Entonces, el espacio total ocupado en todas las cintas de trabajo es, como máximo, la suma de las celdas visitadas en cada cinta, lo cual está acotado por el número total de pasos.

¿Por qué si una MT ocupa espacio $poly(n)$ puede llegar a tardar tiempo $exp(n)$?

Una MT que ocupa espacio $S(n)$ puede tardar mucho más tiempo que $S(n)$. Esto se debe a que la máquina puede realizar muchos pasos dentro del espacio acotado $S(n)$ sin repetir una configuración. Viendo el ejemplo que se presenta en la teoría, si tenemos una MT M con:

- Una cinta de entrada de sólo lectura.
- Una cinta de trabajo.
- $|Q|$ estados.
- $|\Gamma|$ símbolos.
  Podemos calcular la cantidad máxima de pasos, si no loopea como:
  $(n+2) \cdot S(n) \cdot |Q| \cdot |\Gamma|^{S(n)} = c^{S(n)} = exp(S(n))\ pasos$. *Si el espacio $S(n)$ que ocupa la MT es polinomial $poly(n)$, el tiempo máximo de ejecución puede ser exponencial en n $exp(poly(n))$.*

Clase $PSPACE$

A esta clase pertenecen todos los lenguajes que son decididos por una MT en un espacio $poly(n)$.

Propiedades de la clase $PSPACE$

1. Se conjetura que $P \subset PSPACE$.
2. $NP \subseteq PSPACE$
   - Esto se prueba construyendo una máquina de Turing M' que decida si $w \in L$ sin conocer el certificado x. Esto lo podemos hacer recorriendo todos los certificados posibles de longitud $\leq poly(|w|)$. Si bien el tiempo es exponencial ya que la cantidad de certificados es exponencial, podemos hacer que se respete el espacio polinomial con la estrategia de abajo.

Estrategia que seguimos para demostrar que un lenguaje pertenece a $PSPACE$

Una forma de probar esto es usando una estrategia que reutilice el espacio ocupado en una cinta de trabajo, en vez de ir generando más espacio adicional, si no necesitamos almacenar elementos en forma de historial, podemos borrarlos y hacer que un nuevo elemento ocupe ese espacio.

Ejercicio 3

Describir la idea general de una MT M que decida el lenguaje SAT en espacio polinomial. *Ayuda: la generación y la evaluación de una asignación de valores de verdad se pueden efectuar en tiempo polinomial.*

Una máquina de Turing M que decida el lenguaje SAT puede ocupar espacio polinomial si usamos una estrategia que **reúse el espacio de las cintas** y no barra las 2^n **(siendo n la cantidad de variables de** ϕ**)** posibles asignaciones.

Si ϕ tiene n variables, existen 2^n posibles asignaciones, **la máquina en vez de almacenarlas todas las puede simular una por una**, llevando un contador para saber cuál es la asignación que estoy procesando:

1. Establecemos un contador en 1 en una cinta.
2. *Generamos* una asignación A para las n variables de ϕ en otra cinta, esto se puede efectuar en *tiempo polinomial* y, por lo tanto, ocupando *espacio polinomial*.
3. *Verificamos* si la fórmula ϕ es verdadera para la asignación A, esto se puede efectuar en *tiempo polinomial* y, por lo tanto, ocupando *espacio polinomial*.
    - Si es satisfactoria, la máquina se detiene y acepta la fórmula ϕ.
    - Si no es satisfactoria y el contador no es 2^n, se aumenta el contador y se reutiliza el espacio ocupado en el punto 2, volviendo a ejecutar desde ese mismo punto.
    - Si no es satisfactoria y el contador es 2^n, la máquina se detiene y rechaza.

Si bien exploramos una cantidad exponencial de asignaciones posibles, podemos hacer que la MT M **ocupe espacio polinomial.** Usa un contador que en binario mide $O(log_2(2^n))$ que, por propiedades de los logaritmos, termina siendo $O(n)$, más una cinta que mide $O(|\phi|)$. Por lo tanto, $SAT \in PSPACE$.

Clase $LOGSPACE$

La clase $LOGSPACE$ agrupa los lenguajes que pueden ser **decididos** en espacio $O(log_2(n))$.

Los lenguajes de esta clase se consideran **tratables**, esto se debe a que si una MT decide un lenguaje en espacio $S(n) = O(log_2(n))$, entonces el tiempo máximo que puede tardar está acotado por: $T(n) = O(c^{S(n)})$. Si $S(n) = log_2(n)$, entonces: $T(n) = O(c^{log_2(n)})$. Usando la propiedad $a^{log_x(b)} = b^{log_x(a)}$, **esto se convierte en:** $T(n) = O(n^{log_2(c)})$.

Propiedades de la clase $LOGSPACE$

1. Se prueba que $LOGSPACE \subset PSPACE$.
2. Se conjetura que $LOGSPACE \subset P$.

Estrategia que seguimos para demostrar que un lenguaje pertenece a $LOGSPACE$

Una forma de probar esto es usando una estrategia que en vez de almacenar toda la cadena de entrada w en una cinta de trabajo, lo que haga sea utilizar índices para acceder a esa cadena y realizar el trabajo necesario.

Ejercicio 2

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

Una máquina de Turing M que decide el lenguaje $L = \{a^n b^n \mid n \geq 1\}$ en espacio logarítmico podría actuar de la siguiente forma:

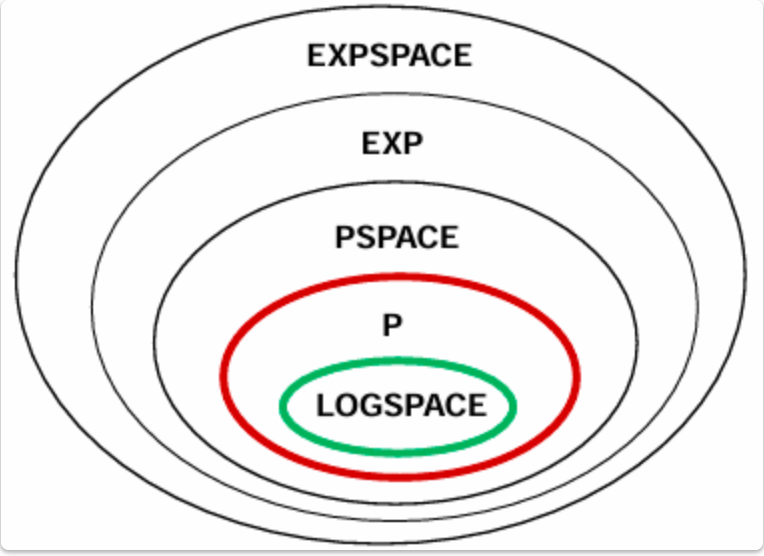
1. Hace $i := 1$ en la cinta 1.
2. Hace $j := n$ en la cinta 2. Si j es impar, rechaza.
3. Copia el símbolo i de w en la cinta 3.
4. Copia el símbolo j de w en la cinta 4.
5. Si $i = (j - 1)$: Si los símbolos son distintos, acepta, si no, rechaza.
   Si $i \neq j$: Si los símbolos son iguales, rechaza.
6. Hace $i := i + 1$ en la cinta 1.
7. Hace $j := j - 1$ en la cinta 2.
8. Vuelve al paso 3.

La MT M ocupa el espacio de los contadores i y j, que en binario miden $O(log_2(n))$, más 2 celdas para alojar cada vez a los símbolos comparados (espacio constante). Por lo tanto, $L \in SPACE(log_2(n))$.

Clase $EXPSPACE$

A esta clase pertenecen todos los lenguajes que son decididos por una MT en un espacio $exp(n)$.

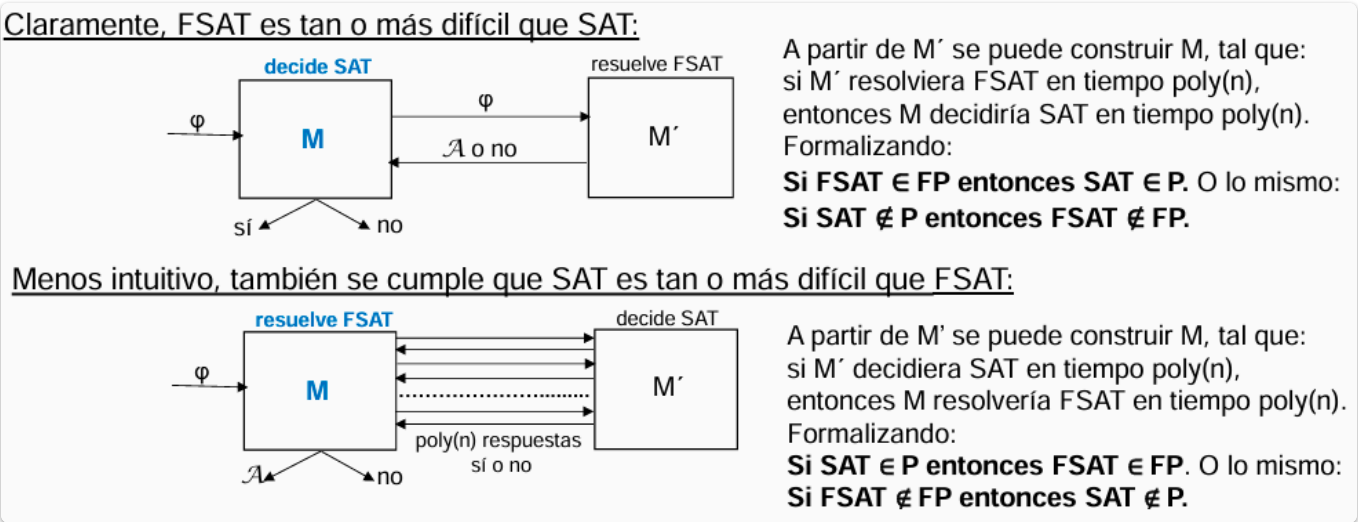
Esquema más completo de la Complejidad Espacial



Misceláneos (Incluyo los tratados en la práctica)

Complejidad temporal de los problemas de búsqueda

Existe una relación general entre un problema de búsqueda FP y su lenguaje asociado (problema de decisión) P, que nos dice que **si un problema de búsqueda FP puede resolverse en tiempo polinomial, también puede decidirse en tiempo polinomial el lenguaje P asociado.** Ejemplo con $FSAT$ y SAT:



MT probabilísticas o MTP

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

Clase BPP (Bounded Probabilistic Polynomial)

Un lenguaje L pertenece a la clase BPP sii existe una MTP M con computaciones de tiempo $poly(n)$ tal que:

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

Propiedades de la clase BPP

1. Se cumple $P \subseteq BPP$ aunque la conjetura más aceptada es que $P = BPP$
   - Si un problema está en P, significa que existe un algoritmo determinista que siempre da la respuesta correcta en tiempo polinomial. Si pasamos un algoritmo determinista de P a un modelo probabilístico (el de BPP), simplemente podemos ignorar el azar, es decir, solo seguimos el algoritmo determinista, y como BPP no obliga a usar la aleatoriedad sino que la permite, $P \subseteq BPP$.

Máquinas cuánticas o MC

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.

Las máquinas cuánticas podrían refutar la Tesis Fuerte de Church-Turing.

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

Clase BQP (Bounded-error Quantum Polynomial Time)

Esta clase contiene los lenguajes aceptados por MC de tiempo $poly(n)$, con probabilidad de error $\leq \frac{1}{3}$.

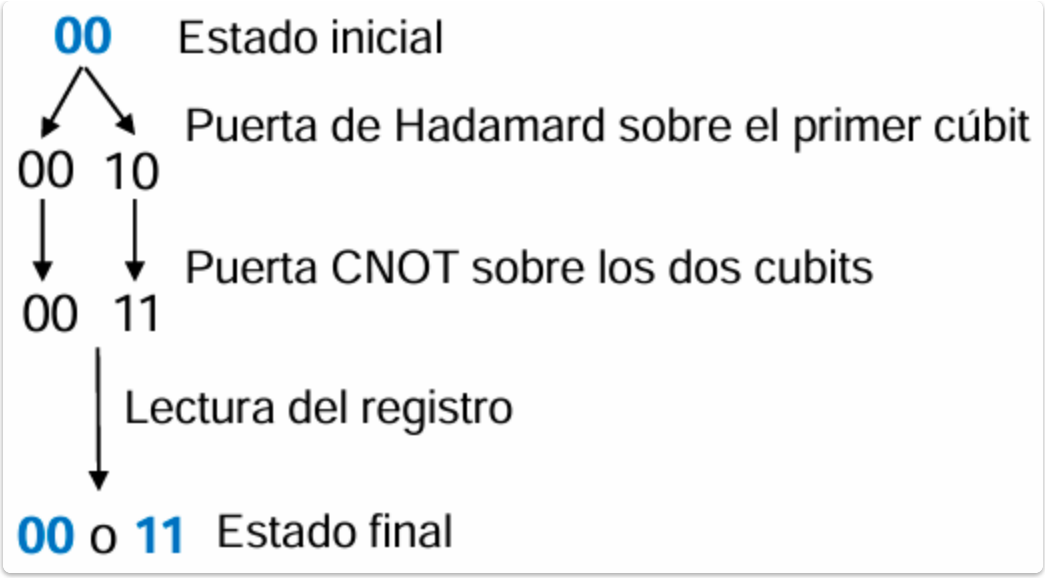
Propiedades de la clase BQP

1. Se cumple que $P \subseteq BQP$ y también $BPP \subseteq BQP$ aunque la conjetura más aceptada es que las dos inclusiones son estrictas.
2. Se acepta la conjetura que BQP y NP son incomparables.
3. Se cumple que $BQP \subseteq PSPACE$.

Ejemplo de puertas cuánticas

Puerta de Hadamard: dado un cúbit 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.


00
Estado inicial
Puerta de Hadamard sobre el primer cúbit
00 10
Puerta CNOT sobre los dos cubits
00 11
Lectura del registro
00 o 11
Estado final