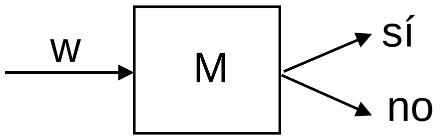
Clase teórica 7

La jerarquía espacio-temporal
y
otros temas de complejidad computacional

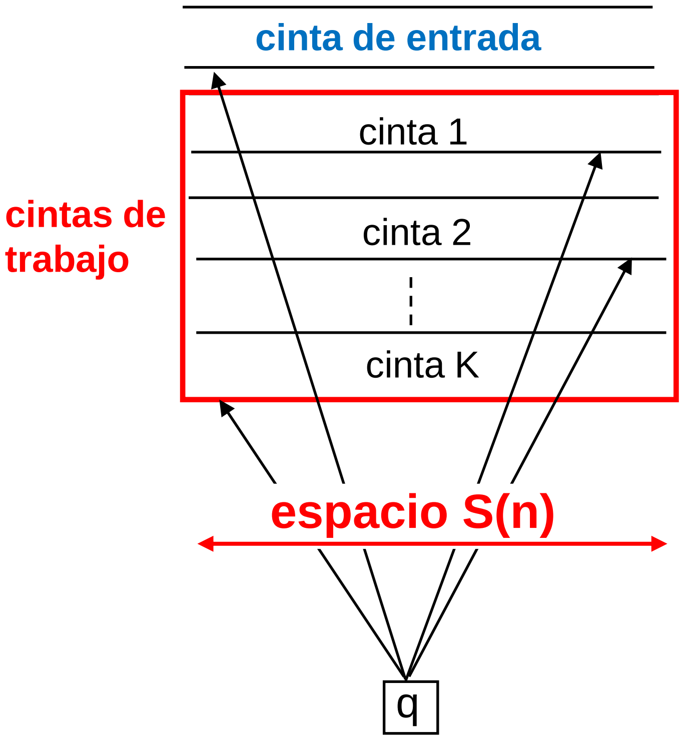
Complejidad espacial



M ocupa a lo sumo S(|w|) celdas, sin considerar la cinta de entrada.

- 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**.

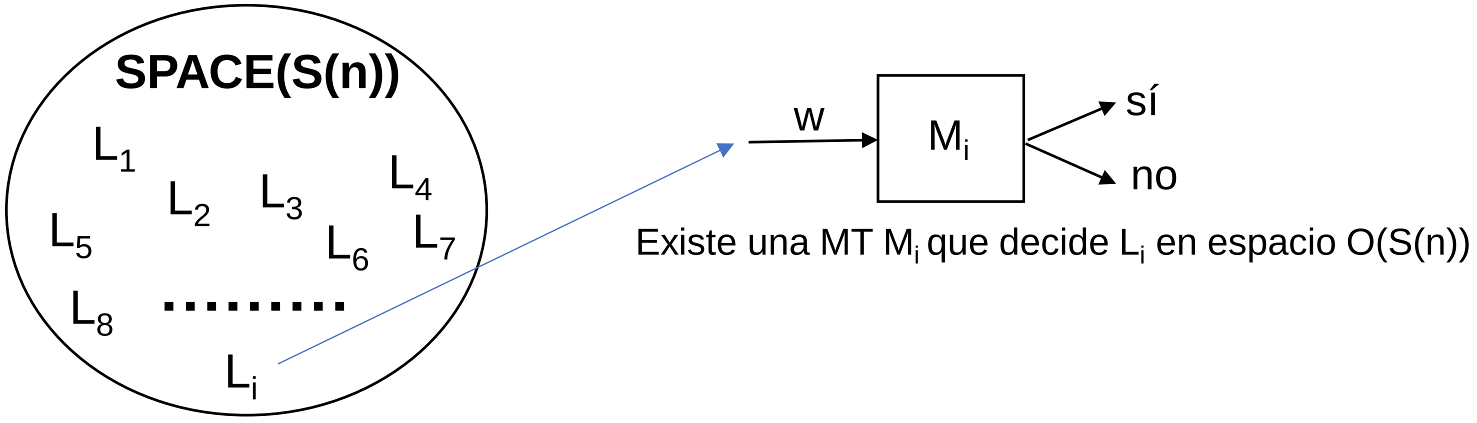
- La cinta de entrada es de **sólo lectura y no se considera en la medición del espacio**. Esto permite un espacio **menor que lineal**, es decir menor que O(n) (la cadena de entrada mide n).



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.

- En el caso más general con una **cinta de salida** se utiliza la misma convención.

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



- Notar que:

1) Una MT M de **tiempo T(n)** ocupa a lo sumo espacio **T(n) celdas**. ¿Por qué?

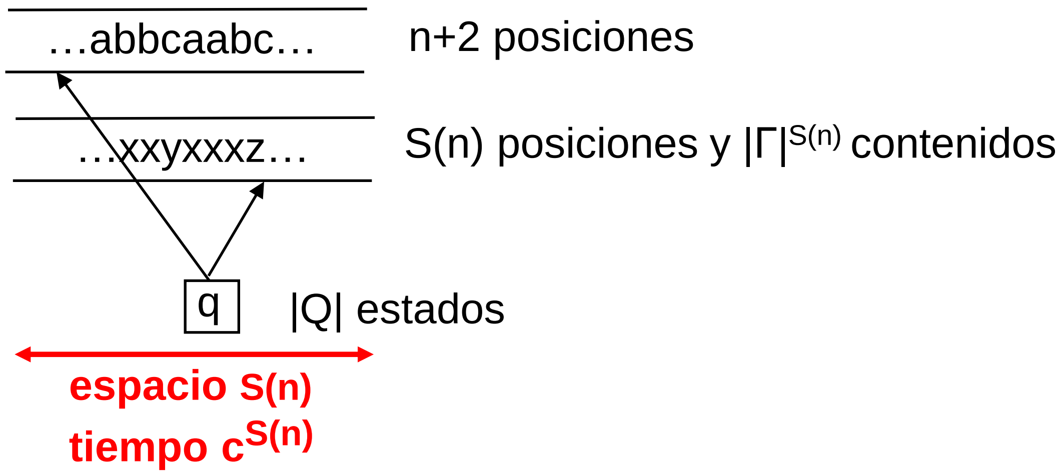
2) En cambio, una MT M que ocupa **espacio S(n)** puede tardar **mucho más que tiempo S(n)**:

Por ejemplo, si M tiene:

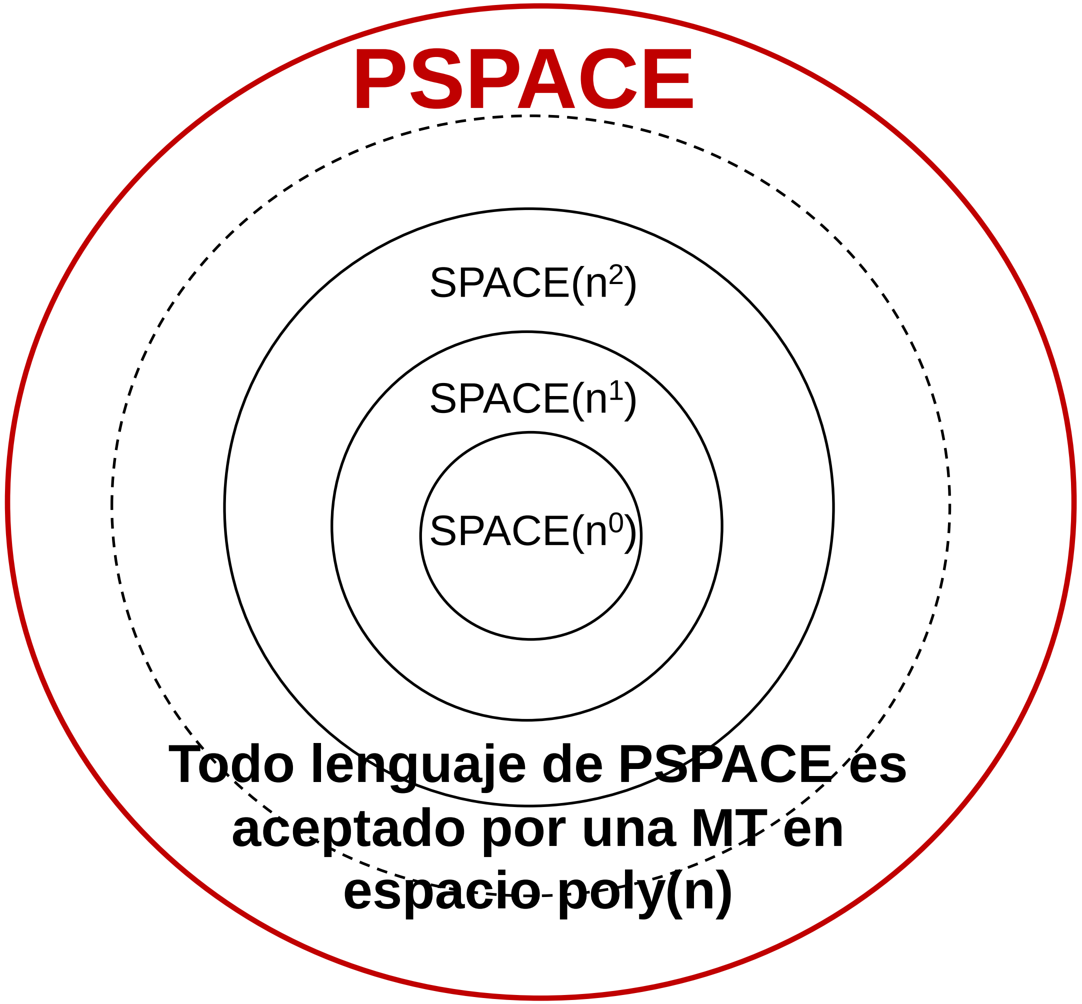
- una cinta de entrada de solo lectura
- una cinta de trabajo
- |Q| estados
- |Γ| símbolos

M puede llegar a hacer, si no loopea:

$(n+2).S(n).|Q|.|\Gamma|^{S(n)}$ $= c^{S(n)}$ = **exp(S(n)) pasos**.



La clase PSPACE



Todo lenguaje de PSPACE es aceptado por una MT en espacio poly(n)

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

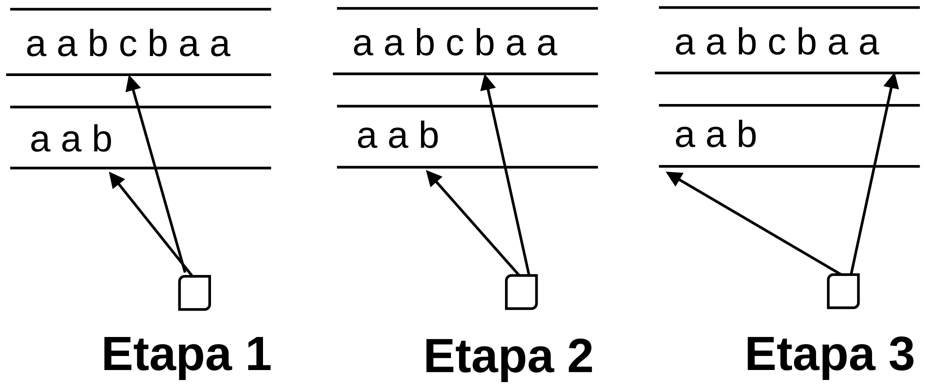
Ejemplo de lenguaje en la clase PSPACE

L = {vcv^R | v^R es la inversa de v y tienen símbolos *a* 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:



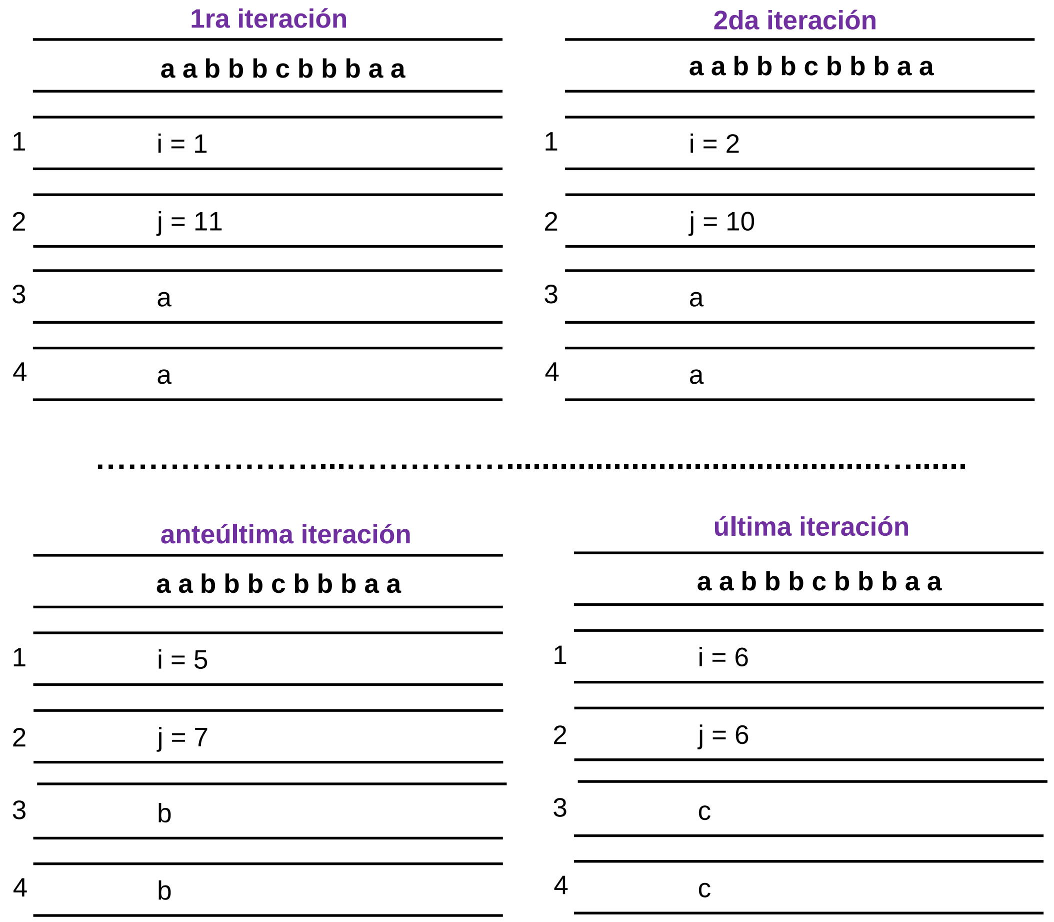
En la cinta 2, M ocupa espacio O(n)

La clase LOGSPACE

- En realidad, **L = {vcv^R | v^R es la inversa de v y tienen símbolos *a* y b}** se puede decidir en menos espacio:

- La siguiente MT decide L en espacio **O(log$_2$n).** Veámoslo por ejemplo con la entrada **w = aabbbcbbbaa**:

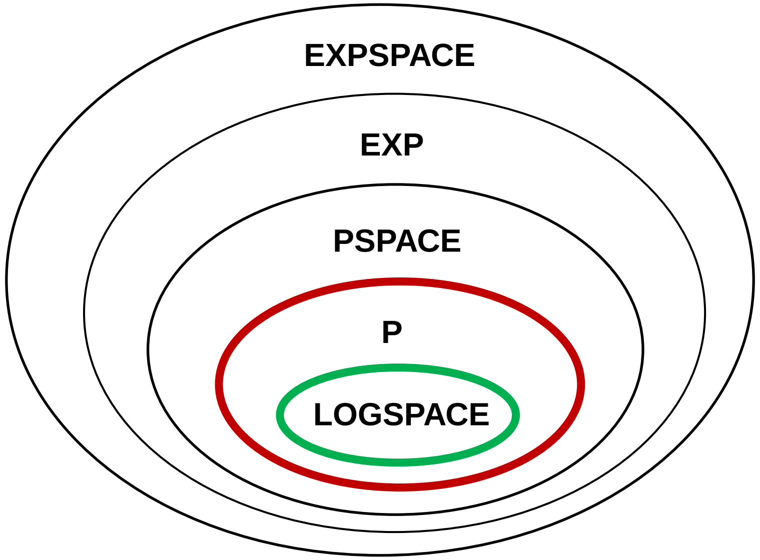
1. Hace i := 1 en la cinta 1.
2. Hace j := n en la cinta 2. Si j es par, rechaza.
3. Copia el símbolo i de w en la cinta 3.
4. Copia el símbolo j de w en la cinta 4.
5. Si i = j: si los símbolos son **c,** acepta, si no, rechaza.
   Si i ≠ j: si los símbolos no son los dos ***a*** o los dos **b**, rechaza.
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 ∈ SPACE(log$_2$n).**
A esta clase se la denomina **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 **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 **poly(n)**

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

Se prueba que LOGSPACE $\subset$ PSPACE

Se conjetura que LOGSPACE $\subset$ P

Se conjetura que P $\subset$ PSPACE

> En definitiva, **LOGSPACE $\subseteq$ P**,
> es decir que **los lenguajes de LOGSPACE son tratables**.

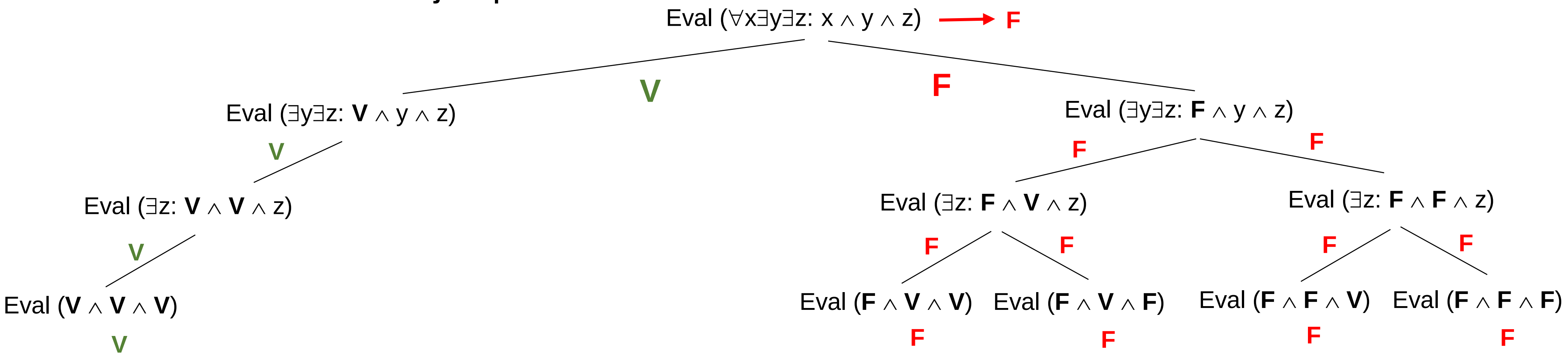
Otro ejemplo de lenguaje en la clase PSPACE

QSAT = {φ | φ 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)

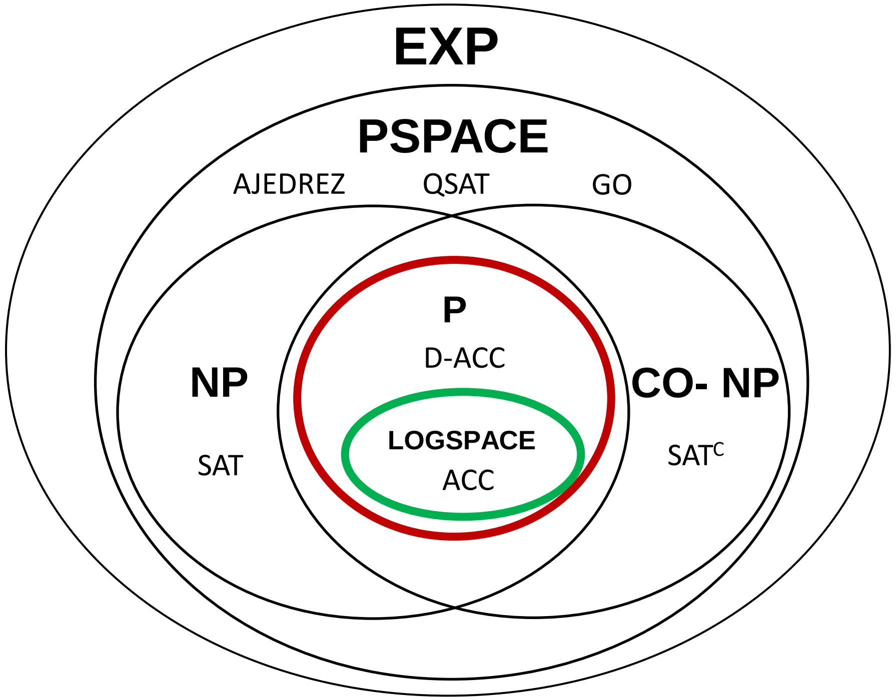
- **QSAT está en PSPACE y no estaría en P ni NP.** La prueba de que QSAT está en PSPACE se basa en una **función recursiva** Eval. Por ejemplo:



- 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(n²).**

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

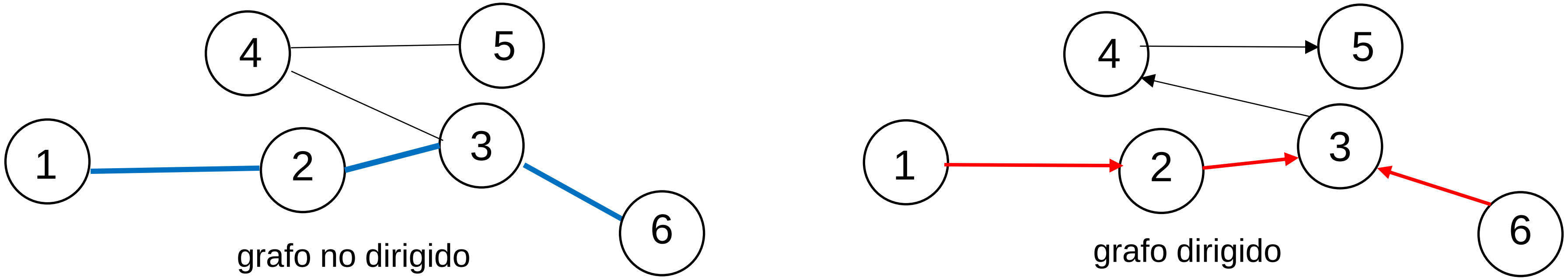
La jerarquía espacio-temporal



- QSAT está entre los lenguajes más difíciles de PSPACE.
  Es **PSPACE-completo**.

- Otros problemas **PSPACE-completos** clásicos se asocian a típicos **juegos entre dos jugadores J_1 y J_2**, y consisten en decidir si J_1 gana: ¿Existe una jugada de J_1 tal que para toda jugada de J_2 existe una jugada de J_1 tal que para toda jugada de J_2 … J_1 gana? Ejemplos: el **ajedrez** y el **go** (adaptados a tableros de n x n).

- Por otro lado,
  ACC = {G | G es un grafo no dirigido con un camino del vértice 1 al vértice m} **está en LOGSPACE,**
  y pareciera que el mismo lenguaje pero definido con grafos dirigidos es más difícil:
  D-ACC = {G | G es un grafo dirigido con un camino del vértice 1 al vértice m} **no estaría en LOGSPACE:**



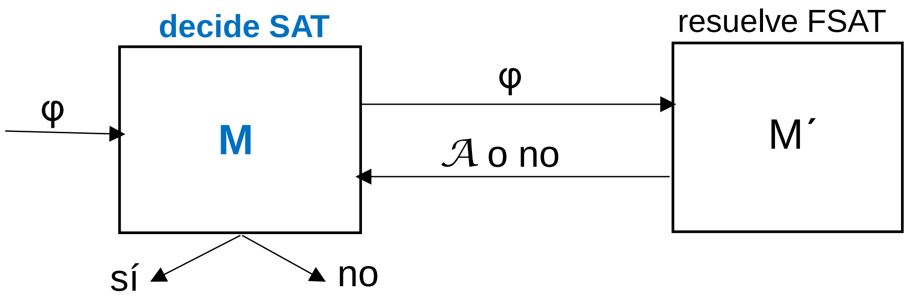
En el segundo caso hay que contemplar la dirección de los arcos

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**.

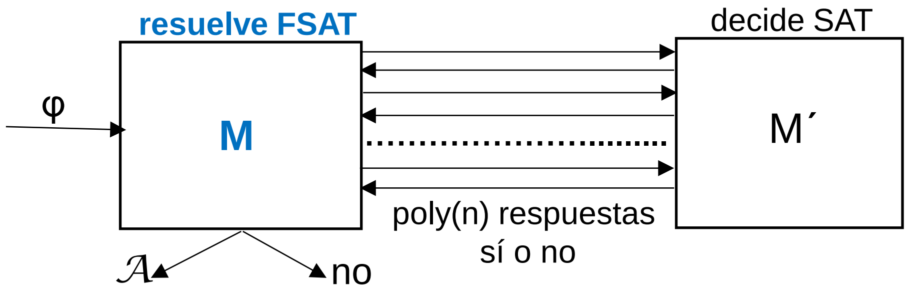
- **Ejemplo. Problemas FSAT (búsqueda) y SAT (decisión).**

<u>Claramente, FSAT es tan o más difícil que SAT:</u>



A partir de M´ se puede construir M, tal que:
si M´ resolviera FSAT en tiempo poly(n),
entonces M decidiría SAT en tiempo poly(n).
Formalizando:
Si FSAT ∈ FP entonces SAT ∈ P. O lo mismo:
Si SAT ∉ P entonces FSAT ∉ FP.

<u>Menos intuitivo, también se cumple que SAT es tan o más difícil que FSAT:</u>



A partir de M' se puede construir M, tal que:
si M´ decidiera SAT en tiempo poly(n),
entonces M resolvería FSAT en tiempo poly(n).
Formalizando:
Si SAT ∈ P entonces FSAT ∈ FP. O lo mismo:
Si FSAT ∉ FP entonces SAT ∉ P.

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

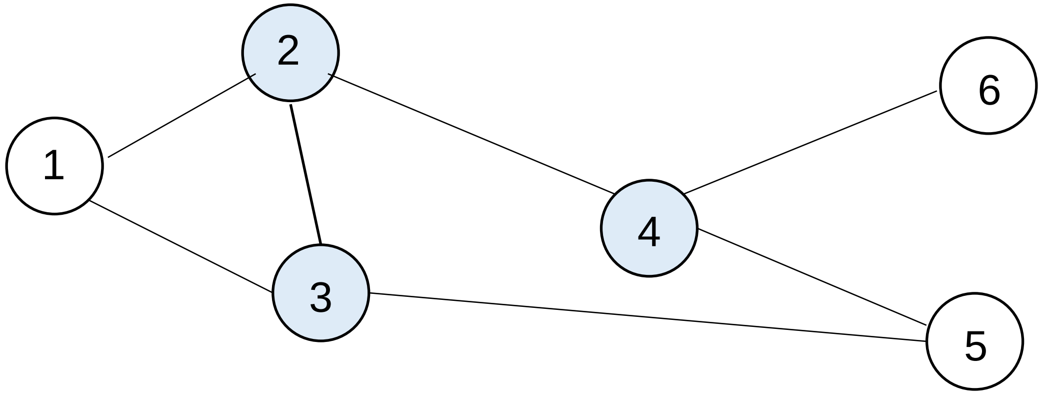
Aproximaciones polinomiales

- 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' := ∅ y E' := E
  2. Si E' = ∅, 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' U $\{v_1, v_2\}$
  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.

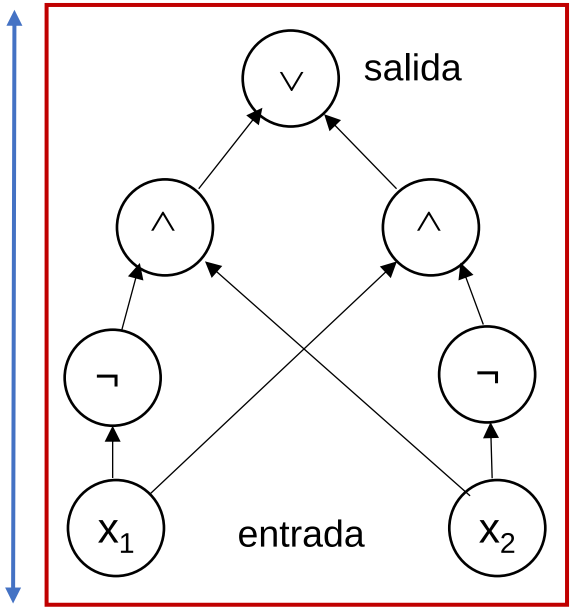
  El cubrimiento mínimo del grafo de arriba tiene **tamaño 3**. La MT M puede generar, entre otros cubrimientos:
  V´ = {1, 2, 3, 5, 4, 6}, de **tamaño 6**, y V´ = {1, 2, 4, 5}, de **tamaño 4** (ejercicio: ¿cuáles otros?).

- Hay problemas con aproximaciones polinomiales muy buenas (**la diferencia Ɛ con el óptimo tiende a 0**), y otros problemas que no cuentan con aproximaciones polinomiales **(cualquier Ɛ que se fije es superado).**

Lenguajes con algoritmos paralelos eficientes

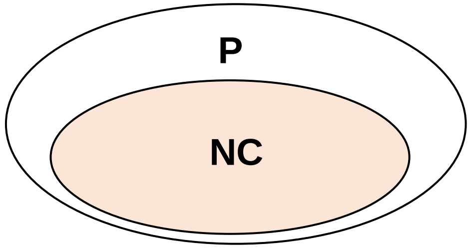
- Modelo más usual de ejecución paralela: **circuito booleano**.
- Un circuito booleano acepta una cadena sii **su salida es 1** (ver ejemplo).
- Para manejar todos los tamaños de cadenas, se usan **familias de circuitos.**
- Se define que un lenguaje cuenta con un **algoritmo paralelo eficiente,** sii lo acepta una familia de circuitos de tamaño **poly(n)** y profundidad **$\log^k$(n).**
- La clase de estos lenguajes se llama **NC** (*Nick class*, por Nicholas Pippenger).
- Problemas clásicos de NC: **operaciones aritméticas, producto de matrices, búsqueda del camino mínimo en un grafo, ordenamiento de un vector, etc.**
- Se cumple **NC $\subseteq$ P**. La conjetura más aceptada es que **NC ≠ P.**

profundidad ("tiempo paralelo")

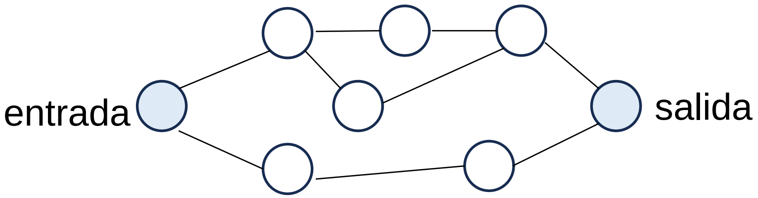


tamaño ("nro de procesadores")

Evaluación de $(x_1 \wedge \neg x_2) \vee (x_2 \wedge \neg x_1)$



- Problema clásico que estaría en P – NC: **máximo flujo en una red**



MT probabilísticas o MTP (algoritmos aleatorios)

Ejemplo 1

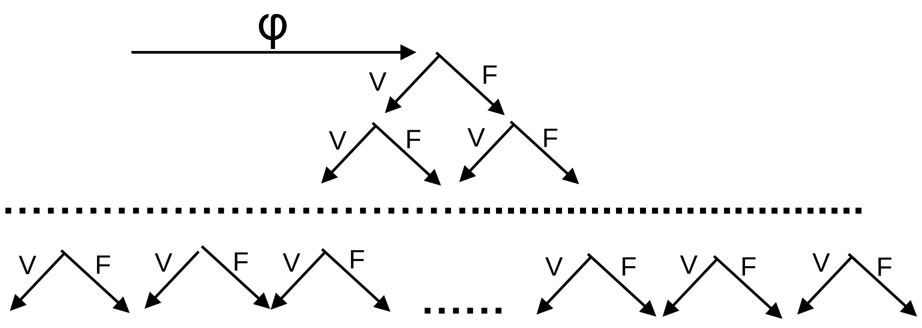
MAYSAT = {φ | φ es una fórmula booleana satisfactible por al menos la mitad más uno de las asignaciones}

Sea la siguiente MTP M. Dada una fórmula φ con m variables:

- Asigna aleatoriamente V o F a la variable x_1.
- Asigna aleatoriamente V o F a la variable x_2.

....

- Asigna aleatoriamente V o F a la variable x_m.
- Evalúa φ con la asignación obtenida y acepta sii φ resulta V.



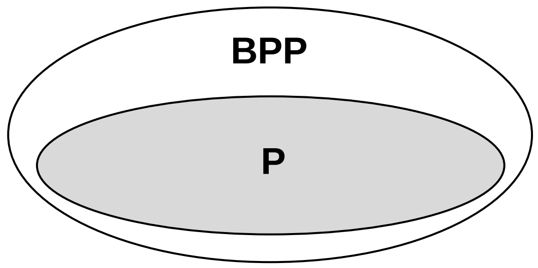
Si φ ∈ MAYSAT, la probabilidad de que M acepte es > 1/2

Si φ ∉ MAYSAT, la probabilidad de que M rechace es ≥ 1/2

La MT **puede equivocarse con cierta probabilidad.**

Definición

- 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*) sii existe una MTP M con computaciones de tiempo poly(n) tal que:
  - Si w ∈ L, M acepta w en **al menos 2/3** de sus computaciones.
  - Si w ∉ L, M rechaza w en **al menos 2/3** de sus computaciones.
- Notar que las MTP asociadas a BPP tienen una **probabilidad de error Ɛ ≤ 1/3.**



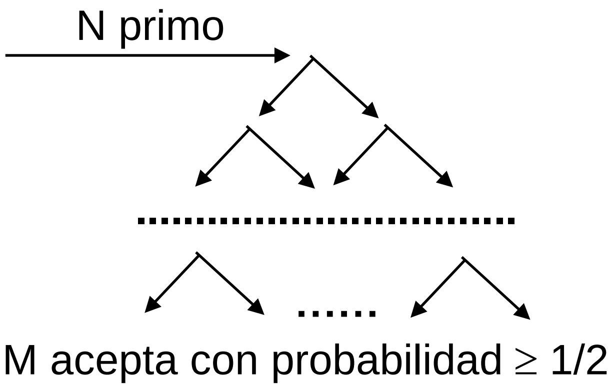
¿Por qué P ⊆ BPP?

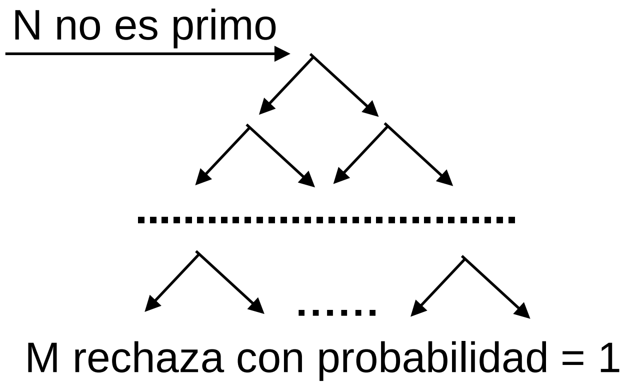
Ejemplo 2

PRIMOS = {N | N es un número primo} pertenece a P, y hasta 2002 sólo se sabía que pertenecía a BPP:

Existe una MTP M que, dada una entrada N:

- Si N ∈ PRIMOS, acepta en **al menos la mitad de sus computaciones.**
- Si N ∉ PRIMOS, 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 $1/2^k$.

- A la clase BPP se la considera **la clase P probabilística.**

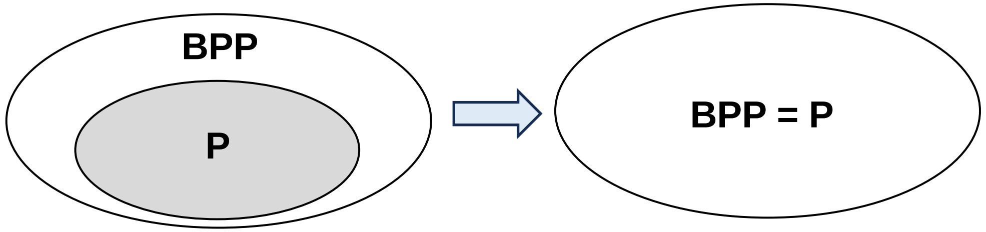
- Existen problemas que hoy día no cuentan con MT eficientes, pero sí MTP eficientes. Por ejemplo, el problema para decidir si **dos polinomios multivariados son iguales.**

¿$5xy^2 + 8zx^4 - 17xyz + 4z = 3x(2y - 13y^4 + 4z) + 2xyz + yx^2$?

- Que PRIMOS esté en P, **no significa que no se use la MTP descripta recién** (es más rápida y simple).

- **P $\subseteq$ BPP** (una MT es un caso particular de MTP, que en todo par de opciones hace lo mismo).

- De todos modos, y sobre todo en los últimos años, **¡la conjetura más aceptada es que P = BPP!**



Es decir, todo algoritmo probabilístico polinomial **se podría desaleatorizar con un retraso 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:

1011 → 11110100001110101110001111101010

semilla aleatoria

Cadena indistinguible de una cadena aleatoria a los ojos de un observador (de tiempo polinomial)

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:

<u>Estados posibles de un cúbit</u>

$|0>$ $|1>$ $\alpha_0|0> + \alpha_1|1>$

tal que, en el último caso, α_0 y α_1 son números complejos que cumplen $\boldsymbol{|\alpha_0|^2 + |\alpha_1|^2 = 1}$**,** que se interpreta así:
- El cúbit está en un estado de superposición de 0 y 1.
- Al **medirlo** (leerlo), se obtiene el valor 0 con **probabilidad** $\boldsymbol{|\alpha_0|^2}$ o el valor 1 con **probabilidad** $\boldsymbol{|\alpha_1|^2}$**.**
- Luego de la medición **se destruye la superposición** (el cúbit queda con el estado básico obtenido).

- Así, utilizando registros con varios cubits, la capacidad computacional de las MC se torna **muy grande**:

1 cúbit guarda la misma información que 2 bits clásicos: 0 y 1.
2 cubits guardan la misma información que 4 pares de bits clásicos: 00, 01, 10 y 11.
3 cubits guardan la misma información que 8 ternas de bits clásicos: 000, 001, 010, …
Etc.

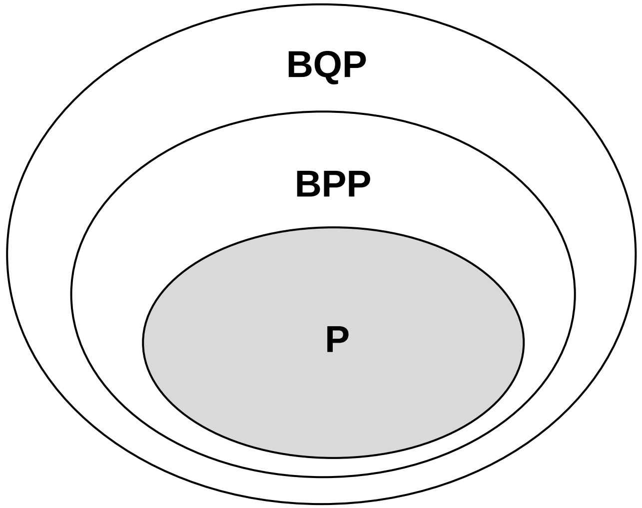
- Considerando por ej. una MC con un registro de 3 cubits: la MC procesa "en paralelo" los 8 estados superpuestos, hasta que se mide el registro, luego de lo cual se obtiene el estado s_i con probabilidad $|\alpha_i|^2$ y se pierde la superposición.

Ej. con 3 cubits

α_0	000
α_1	001
α_2	010
α_3	011
α_4	100
α_5	101
α_6	110
α_7	111

- 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 **poly(n)**, con **probabilidad de error ≤ 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.



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

- Se cumple **P** $\subseteq$ **BQP** y también **BPP** $\subseteq$ **BQP**.
  La conjetura 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.**

- Un ejemplo de problema en BQP que hoy día no tiene un algoritmo clásico polinomial es la **factorización**.
  El algoritmo cuántico de **Shor** resuelve la factorización en tiempo polinomial.

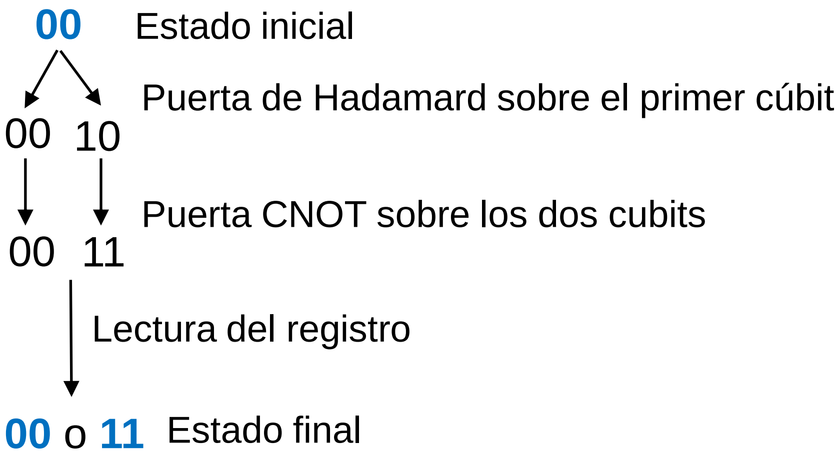
- Las MC **no podrían decidir eficientemente los lenguajes NP-completos.**
  El algoritmo de **Grover** acelera "sólo" cuadráticamente el mejor algoritmo clásico para resolver SAT.

- Las MC en principio son factibles físicamente. Un aspecto a considerar muy relevante es el **"ruido" (interferencias),** que atenta contra las superposiciones de estados. Para combatirlo se vienen desarrollando distintos **procedimientos de manejo de errores.**

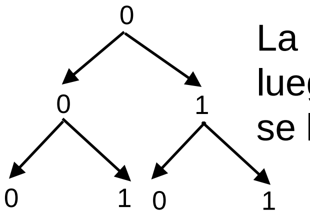
Ejemplo de puertas cuánticas y computación cuántica

- *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.**

1. Sea un registro cuántico de dos cubits en el estado 00.
2. Aplicando sobre el primer cúbit la *puerta de Hadamard*, se obtiene la supersposición de los estados 00 y 10.
3. Aplicando sobre los dos cubits la *puerta CNOT*, se obtiene la superposición de los estados 00 y 11.
4. Leyendo el registro, se obtiene el estado 00 o el estado 11, cada uno con probabilidad 1/2, y se destruye la superposición cuántica.

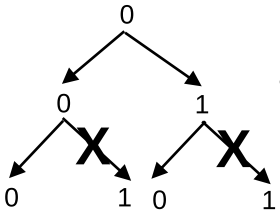


Diferencia entre los algoritmos probabilísticos y los algoritmos cuánticos



algoritmo probabilístico

La probabilidad de que luego de 2 tiros de moneda se lea la cadena 11 es 1/4.

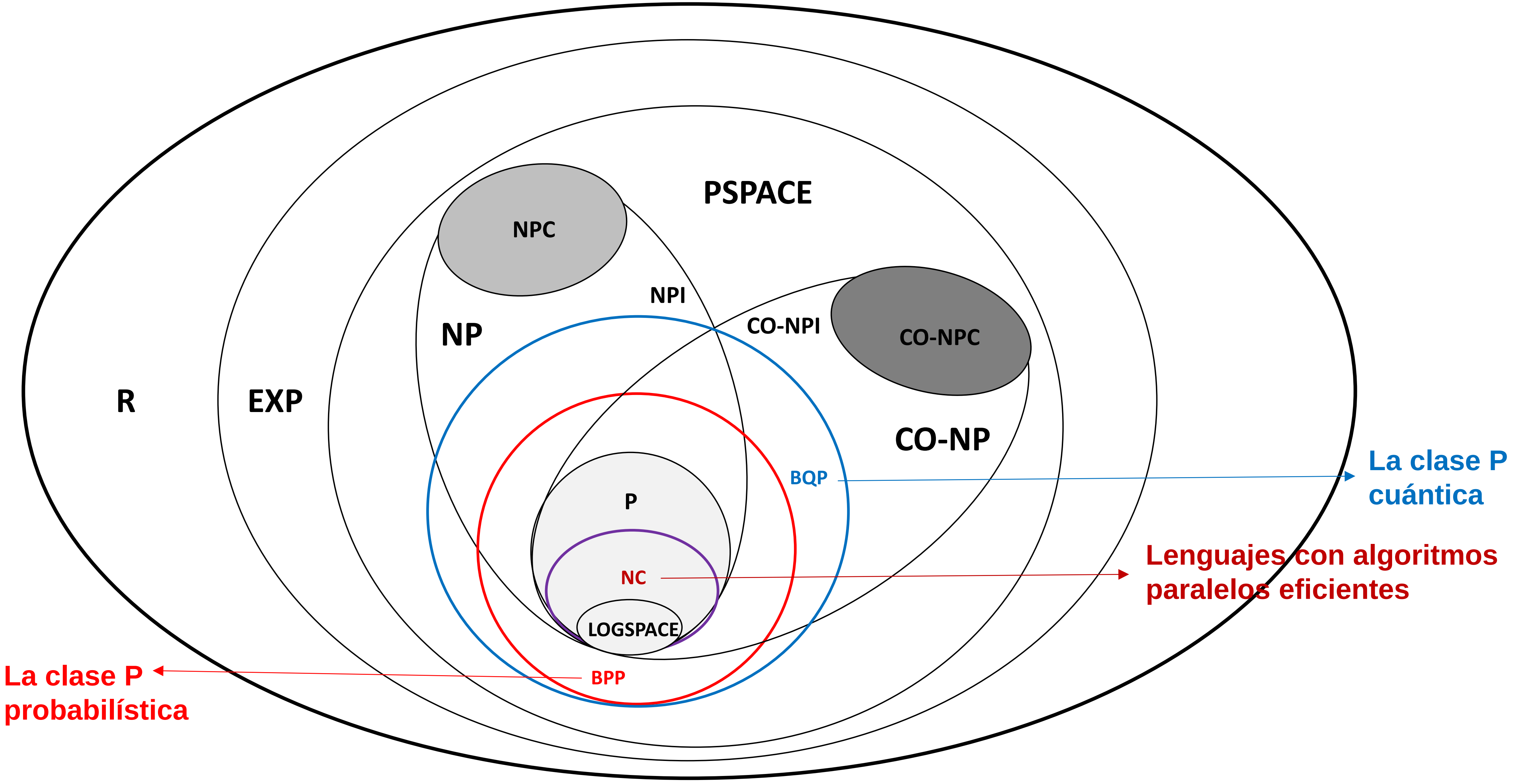


algoritmo cuántico

La probabilidad de que luego de 2 aplicaciones de la puerta cuántica de Hadamard se lea la cadena 11 es 0.

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

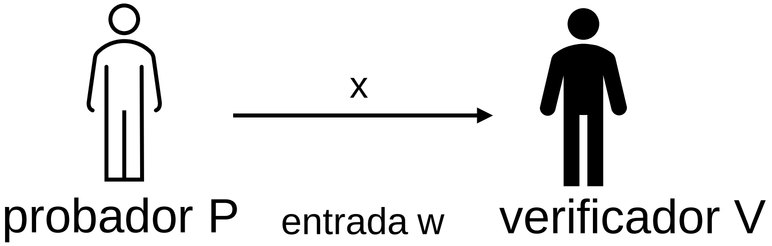
Las distintas jerarquías descriptas



Anexo

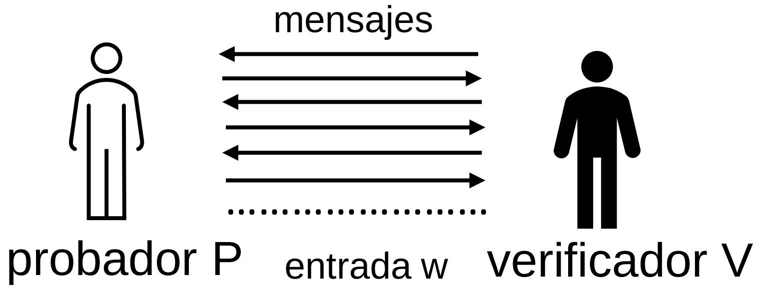
Sistemas interactivos

- **Todo lenguaje L de NP cuenta con un verificador eficiente (MT de tiempo polinomial):**



- Si $w \in L$, entonces existe un probador P que puede convencer al verificador V de que $w \in L$, por medio de un certificado sucinto x.
- Si $w \notin L$, entonces no existe ningún probador P que pueda convencer al verificador V de lo contrario, utilice el certificado x que utilice.
- Los probadores P son MT de poder ilimitado.

- **El mismo esquema se puede generalizar a una serie de interacciones entre P y V:**



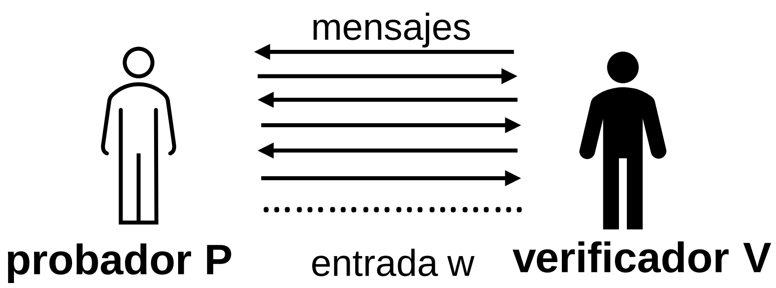
- El verificador V y el probador P se intercambian poly(|w|) mensajes de tamaño poly(|w|) (V envía preguntas y P envía respuestas).
- En este caso, el certificado x incluye todos los intercambios entre P y V.

- ¿Qué sucede cuando V es una MT probabilística? **¿Nos mantenemos dentro de la clase NP?**

La respuesta es **NO.** Un sistema interactivo de este tipo amplía su alcance a **toda la clase PSPACE.**

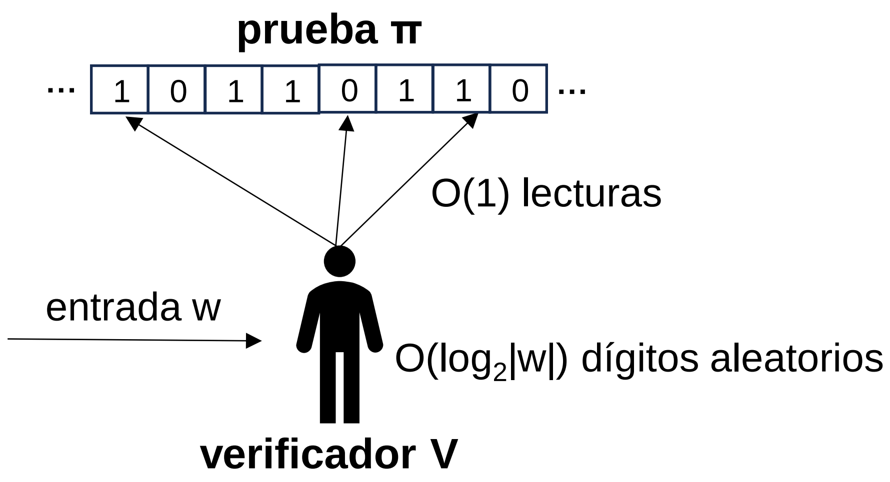
Otras caracterizaciones de NP con sistemas interactivos

- Con ciertas asunciones, se prueba que todo lenguaje L de NP cuenta con un sistema interactivo en el que V **no aprende nada** de P (sólo la pertenencia de w a L), conocido como **prueba de conocimiento cero**.



- En una prueba de conocimiento cero, el verificador V no aprende nada de sus interacciones con el probador P (**esquema muy útil en los procesos de seguridad, p.ej. en las autenticaciones**).

- Se prueba también que todo lenguaje L de NP cuenta con un **sistema de pruebas chequeables probabilísticamente** de tipo **PCP($\log_2$n, 1):**



- En este caso no hay un probador. Hay una prueba π de una cadena w, prueba a la que V accede un **número constante de veces**, usando **$\log_2$|w| dígitos aleatorios de la prueba π.**

- ¡Esto significa que es tan seguro revisar una prueba línea por línea, como se hace habitualmente, como leer apenas unos pocos bits de la misma! **(esta propiedad es útil en el área de las aproximaciones polinomiales).**

Clase práctica 7

Ejemplo. Prueba de la NP-completitud del problema del cubrimiento de vértices

- Probaremos que CV = {(G, K) | G es un grafo no dirigido que tiene un cubrimiento de vértices de tamaño K} es NP-completo.

- Proponemos la siguiente reducción polinomial de 3-SAT (NP-completo) a CV (se prueba fácil que está en NP):

  f(φ) = (G, 2K)

  tal que φ es una fórmula booleana en FNC con tres literales por cláusula, K es la cantidad de cláusulas de φ, y G es un grafo que se construye de la siguiente forma:

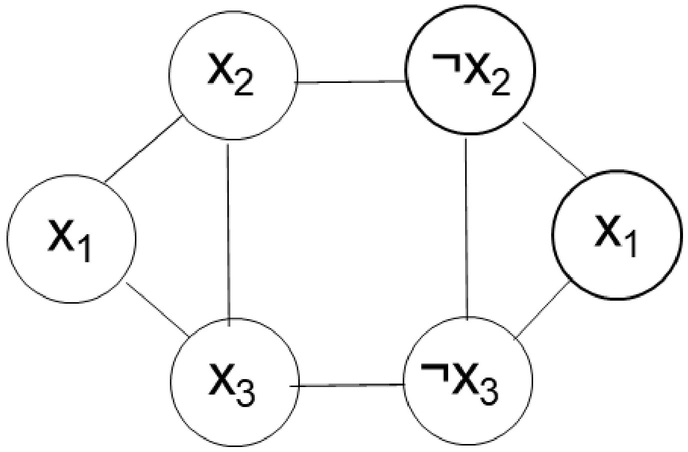
  1. Por cada literal de cada cláusula de φ se crea un vértice en G.

  2. Todo par de vértices creados a partir de dos literales de una misma cláusula se unen por un arco (identificado como *enlace de tipo 1*). De esta manera, toda cláusula determina un triángulo con enlaces de tipo 1.

  3. Todo par de vértices creados a partir de dos literales x_i y $\neg x_i$ de distintas cláusulas también se unen por un arco (identificado como *enlace de tipo 2*).

(sigue)

- Por ejemplo, el grafo G siguiente se obtiene a partir de la fórmula booleana $\boldsymbol{\varphi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)}$:



- Notar que $\boldsymbol{\{x_2, x_3, \neg x_2, \neg x_3\}}$ es un **cubrimiento de vértices de tamaño 4 de G,** el doble de la cantidad de cláusulas de φ, tal como se requiere.

- La función f es efectivamente una **reducción polinomial de 3-SAT a CV.** En tiempo polinomial, por cada una de las K cláusulas se puede crear un triángulo, y por cada literal de cada cláusula se pueden crear O(K) arcos.

- Y se cumple $\boldsymbol{\varphi \in}$ **3-SAT sii (G, 2K)** $\boldsymbol{\in}$ **CV:**

(sigue)

- **Sean φ ∈ 3-SAT, $\mathcal{A}$ una asignación que satisface φ, y el siguiente conjunto C de 2K vértices de G:**

  Incluye a todos los vértices asociados a los literales falsos de φ según $\mathcal{A}$, y si con esto no es suficiente para que tenga dos vértices de cada triángulo (por existir cláusulas de φ con dos o tres literales verdaderos según $\mathcal{A}$), incluye más vértices hasta completar el par.

  Se cumple que C es un cubrimiento de vértices de G: todo enlace de tipo 1 está cubierto porque C tiene dos vértices de cada triángulo formado con dichos enlaces, y todo enlace de tipo 2 también está cubierto porque uno de sus vértices está asociado a un literal falso según $\mathcal{A}$, el cual está en C.

  Así, (G, 2K) ∈ CV.

- **Sean (G, 2K) ∈ CV, C un cubrimiento de vértices de tamaño 2K de G, y la siguiente asignación $\mathcal{A}$ a φ:**

  A los literales asociados a los vértices que no están en C les asigna el valor *verdadero*, y al resto de los literales les asigna valores consistentes cualesquiera.

  Se cumple que $\mathcal{A}$ satisface φ: C está compuesto necesariamente por dos vértices de cada uno de los triángulos, y por lo tanto en toda cláusula hay un literal verdadero según $\mathcal{A}$. Además, $\mathcal{A}$ no tiene inconsistencias, porque si dos literales x_i y $\neg x_i$ de dos cláusulas distintas son verdaderos según $\mathcal{A}$, significa que el enlace de tipo 2 asociado no está cubierto por C (contradicción).

  Así, φ ∈ 3-SAT.