Teoremas de Jerarquía

Seccion 9.1 Indroducción a la Teoría de la Computación, Michael Sipser

ANA SOFÍA HERNÁNDEZ ZAVALA, Universidad Nacional Autónoma de México, Facultad de Ciencias, México
SANLUIS CASTILLO DANIELA ALEJANDRA, Universidad Nacional Autónoma de México, Facultad de Ciencias, México
METZITLALLI ÁLVAREZ RÍOS, Universidad Nacional Autónoma de México, Facultad de Ciencias, México

Este es el resumen de la investigación. (Ya puedes borrar el comentario confuso que estaba aquí).

CCS Concepts: • **Networks** → **Network reliability**.

Additional Key Words and Phrases: teoremas, jerarquías, tiempo, espacio, corolario, definicion, máquina de Turing

ACM Reference Format:
Ana Sofía Hernández Zavala, Sanluis Castillo Daniela Alejandra, and Metzitlalli Álvarez Ríos. 2025. Teoremas de Jerarquía: Seccion 9.1 Indroducción a la Teoría de la Computación, Michael Sipser. *ACM Trans. Storage* 1, 1, Article 1 (November 2025), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introducción

El sentido común nos dice que si le damos más tiempo o más espacio a una máquina de Turing entonces debería de incrementar la clase de problemas que podría resolver; y los Teoremas de Jerarquía lo confirman, ya que estos teoremas prueban que las clases de complejidad de tiempo y espacio no son todas las mismas.
Por ejemplo en este articulo mostraremos que el teorema de jerarquía de complejidad del espacio es más simple que el del tiempo.

2 Preliminares

A continuación recordamos las definiciones de completitud para clases de complejidad exponencial, necesarias para estudiar el problema $EQ_{REX\uparrow}$.

Definition 2.1. Un lenguaje B es **EXPSPACE-completo** si:[3, p. 344]

(1) $B \in$ EXPSPACE, y
(2) todo lenguaje $A \in$ EXPSPACE es reducible en tiempo polinómico a B.

Definition 2.2. Análogamente, un lenguaje B es **EXPTIME-completo** si:[1, p. 2]

Authors' Contact Information: Ana Sofía Hernández Zavala, Universidad Nacional Autónoma de México, Facultad de Ciencias, Ciudad de México, México, anasofiahdzz@ciencias.unam.mx; Sanluis Castillo Daniela Alejandra, Universidad Nacional Autónoma de México, Facultad de Ciencias, Ciudad de México, México, danielasanluisc@ciencias.unam.mx; Metzitlalli Álvarez Ríos, Universidad Nacional Autónoma de México, Facultad de Ciencias, Ciudad de México, México, metzi.ar18@ciencias.unam.mx.

ACM 1553-3093/2025/11-ART1
https://doi.org/10.1145/nnnnnnn.nnnnnnn

(1) $B \in$ EXPTIME, y
(2) todo lenguaje $A \in$ EXPTIME se reduce en tiempo polinómico a B.

3 Teorema del Espacio

Definition 3.1. Una función $f : N \longrightarrow N$, donde $f(n)$ es al menos $O(log_n)$, es llamada espacio constructible si la funcion que mapea la cadena de 1^n a la representación binaria de $f(n)$ es computada en espacio $O(f(n))$. [?]

Es decir que f es un espacio constructible si alguna máquina de Turing M de tiempo $O(f(n))$ existe y siempre se detiene con la representacion binaria de $f(n)$ en su cinta cuando empieza en la entrada 1^n. [?]. Se encontró una mejora significativa.
El rol del espacio constructible en el teorema de jerarquía del espacio se entiende mejor de la siguiente manera: Si se tiene un $f(n)$ que es mayor o un poco más grande que $g(n)$ en espacio, entonces $f(n)$ debería de poder analizar más lenguajes, pero supongamos que $f(n)$ esta analizando un lenguaje muy grande entonces ocupa todo el espacio sobrante o incluso requerirá más espacio del diponible.
Es así como llegamos al teorema formal de la jerarquía del espacio.

Theorem 3.2. *Para cualquier función $f : N \longrightarrow N$ del espacio constructible, existe un lenguaje A que es decidible en espacio $O(f(n))$ pero no en espacio $o(f(n))$. [?]*

La prueba a lo anterior es básicamente demostrar que el lenguaje A tiene 2 propiedades:

- A es decidible en espacio $O(f(n))$.
- A no es decidible en espacio $o(f(n))$.

Describiendo a A con un algoritmo D que lo decide, D corre en espacio $O(f(n))$, así cumple con la primer propiedad; y D garantiza que A es diferente de cualquier lenguaje que es decidible en espacio $o(f(n))$, lo cual asegura la segunda propiedad.
Esto dado a

Corollary 3.3. *Para cualesquiera 2 funciones $f_1, f_2 : N \longrightarrow N$, donde $f_1(n)$ es $o(f_2(n))$ y f_2 es espacio constructible, $SPACE(f_1(n)) \subsetneq SPACE(f_2(n))$. [?]*

En otras palabras, $SPACE(o(f(n))) \subsetneq SPACE(f(n))$, siendo *SPACE(o(f(n)))= {B | alguna maquina de Turing M que decide B en espacio o(f(n))}* [2]

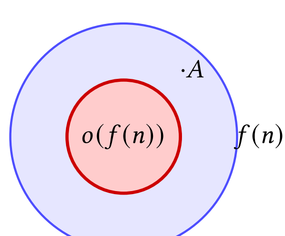

Fig. 1. $SPACE(o(f(n))) \subsetneq SPACE(f(n))$.

Parecido a la situación de un lenguaje libre de contexto, en el caso de los lenguajes regulares, donde se muestra un lenguaje particular que se diferencia por ser libre de contexto pero no regular. Usando la prueba de diagonalización, se construye una máquina de Turing D que decide el lenguaje A con las siguientes propiedades:

(1) D generará el lenguaje A.
(2) D se ejecutará dentro de $f(n)$.
(3) D se diseñará para asegurarse que su lenguaje no pueda implementarse en un espacio menor, para eso se asegura que su lenguaje sea diferente de cualquier lenguaje decidible por una máquina de Turing en un espacio menor.
(4) D se asegurará que no puede ser implementada en $o(f(n))$.

Prueba: Dada una máquina de Turing D donde:

(1) D corre en espacio $O(f(n))$.
(2) D es cierto que $L(D) \neq L(M)$ para cualquier MT M que corra en espacio $o(f(n))$.
(3) Dejar $A = L(D)$.

El objetivo de esto es mostrar que $A \in SPACE(f(n))$ pero $A \notin SPACE(o(f(n)))$.

(1) D recibe como entrada w
(2) Marcar las celdas de la cinta Problema$f(n)$ donde $n = |w|$; si trata de usar más cinta, rechaza (solo se permitirá que use el espacio $f(n)$ de lo contrario tal vez D no este en $f(n)$). Para asegurarnos de que eso no pase entonces se coloca el # para delimitar el espacio $f(n)$, si trata de usar más que eso, rechaza.
(3) Si $w \neq< M >$ para alguna máquina de Turing M, rechazar (w no describe nada, solo es un salto). Rechaza a menos que si sea una w que describa una maquina de Turing M.
(4) Simular * M en w.
Acepta si M rechaza.
Rechaza si M acepta.
*NOTA: D puede simular M con un factor constante de espacio.

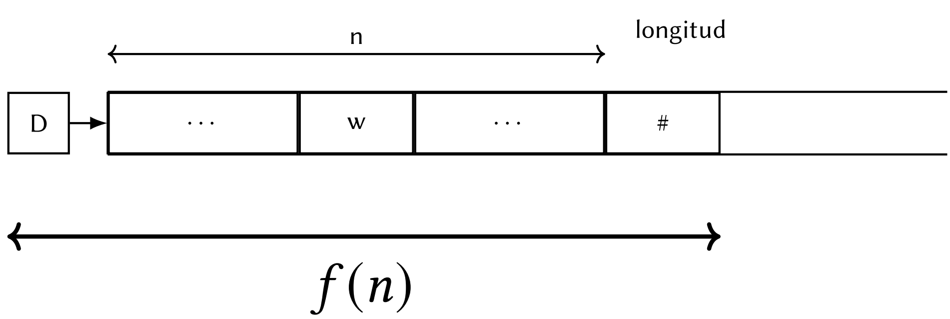

Fig. 2. Punto 1. Solo se permitirá que D use el espacio f(n), de lo contrario tal vez D no este en f(n).

D está haciendo algo diferente a A, D no puede ser diferente de cada M porque D en sí misma es una máquina de Turing, D solo se ejecuta dentro de celdas $f(n)$ de la cinta, tiene que poder realizar esa simulación de M dentro de esa cantidad de cinta, siempre rechazará si usa más.
Si M usa menos que D, entonces puede hacer la simulación.

3.1 Problemas

3.1.1 ¿Qué pasa si M corre en tiempo o(f(n)) pero tiene una constante grande? Entonces D no tendrá espacio para simular M cuando es pequeña.
Solución: Simular M en infinitos $w's$. Pensando en w como la representación de M pero con un número ilimitado de ceros finales. Se cambia el punto 3. por $3.Siw \neq< M > 10^*$ por alguna máquina de Turing M, rechaza.

Lo primero que se hará con w es eliminar los ceros finales hasta el último 1 y tomar el resto como la descripción de la máquina. Si M de verdad esta corriendo en $o(f(n))$ entoces habrá espacio suficiente para que M se ejecute completamente sobre w y así diferenciarlo de él.

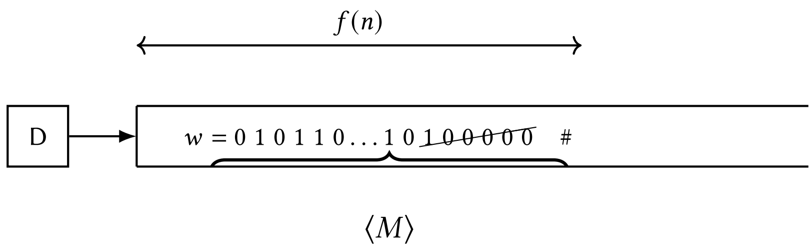

Fig. 3. Problema 2.2.1 <M> abarca desde el primer 0 hasta el último 0, y se debe de tachar desde el último 0 hasta el último 1.

Ahora M se ejecuta en una gran entrada, suficientemente grande para que D (que tiene más espacios) pueda ejecutarse completamente vacía.

3.1.2 ¿Qué pasa si M se cicla? D debe detenerse. Solución: Detener M si corre en $2^{f(n)}$ pasos. *Modificando el paso 4. por 4. Simular * M en w por $2^{f(n)}$ pasos. Acepta si M rechaza. Rechaza si M acepta o no se ha detenido.

3.1.3 ¿Cómo computar f? f tiene que ser computable dentro del espacio.
Solución: Asumir que f es un espacio constructible, i.e. puede computar f con $O(f(n))$. Ciertas funciones como $log_n, log_n^2, n, n^2, 2^n, \ldots$ son espacios constructibles.
¿Se puede decir que D tiene como entrada M y simula M sobre sí mismo? Cierto.

4 Teorema del Tiempo

Definition 4.1. Sea $t : \mathbb{N} \longrightarrow \mathbb{N}$ una función construible en tiempo. Entonces existe un lenguaje A tal que es decidible en tiempo $O(t(n))$ pero no es decidible en tiempo $o\left(\frac{t(n)}{\log t(n)}\right)$. [3, p. 341]

El teorema de jerarquía temporal establece que, bajo condiciones razonables, disponer de más tiempo permite decidir más lenguajes. De manera precisa, si $t(n)$ es una función construible en tiempo, entonces existe un lenguaje que puede ser decidido en tiempo $O(t(n))$, pero que no puede ser decidido en tiempo $o\left(\dfrac{t(n)}{\log t(n)}\right)$.

Para demostrar el teorema, se define una MT D diseñada para "diferenciarse" de todas las máquinas que corren en tiempo más pequeño que $t(n)/\log t(n)$. [3, p. 341] La idea es la siguiente:

(1) D recibe como entrada una cadena de la forma
$$w = \langle M \rangle 10^k,$$
es decir, codificación de una máquina M seguida de un 1 y varios ceros.
(2) D calcula el valor $t(n)$, donde $n = |w|$, utilizando el hecho de que $t(n)$ es construible.
(3) D simula a la máquina M sobre la misma entrada w, con límite de tiempo igual a $t(n)$.
(4) Cuando la simulación termina antes de agotar el tiempo, D responde lo opuesto a lo que responde M:
- si M acepta, D rechaza;

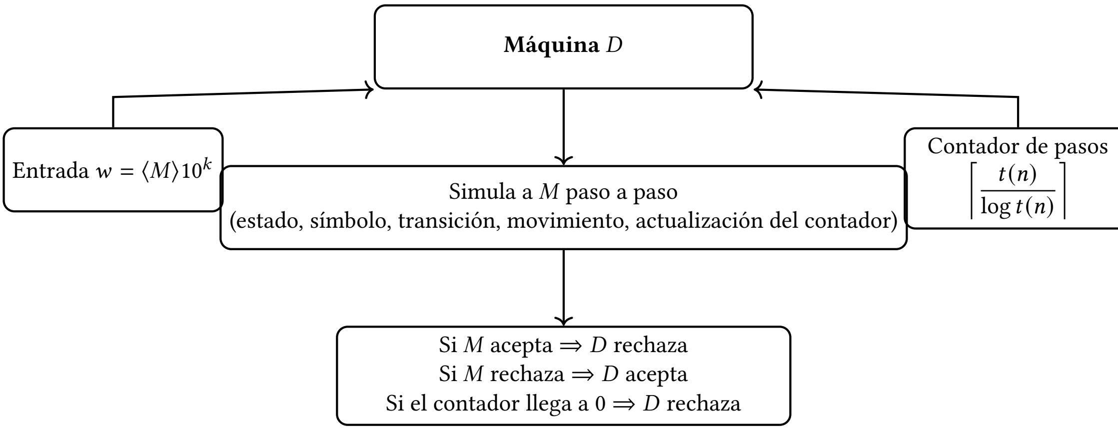

Fig. 4. Esquema de la simulación temporal realizada por la máquina D. La máquina calcula $t(n)$, establece un contador de pasos y simula a M sin exceder el tiempo permitido.

- si M rechaza, D acepta.

Esta técnica es un ejemplo clásico de *diagonalización*.

La diferencia entre el teorema de tiempo y el de espacio aparece cuando analizamos el costo de la simulación.

Simular un solo paso de una máquina M requiere que D:

- Lea el estado actual de M, consulte el símbolo bajo la cabeza,busque la transición correcta, escriba el nuevo símbolo, mueva la cabeza, y actualice un contador de pasos disponibles.

Este contador debe almacenar un número de tamaño aproximadamente $\log t(n)$, y cada actualización requiere tiempo proporcional a ese tamaño. Por lo tanto, cada paso simulado aporta un costo adicional de:

$$O(\log t(n)).$$

Como consecuencia, para que la simulación complete al menos $g(n)$ pasos de M, debe cumplirse:

$$g(n) \cdot \log t(n) \leq t(n).$$

Esto implica:

$$g(n) \leq \frac{t(n)}{\log t(n)}.$$

Esa es la razón exacta por la cual el resultado establece una separación entre:

$$\mathrm{TIME}(t(n)) \qquad \text{y} \qquad \mathrm{TIME}\left(o\left(\frac{t(n)}{\log t(n)}\right)\right).$$

Corollary 4.2 (Corolario 9.11). *[3, p. 343]*
Para cualesquiera dos funciones $t_1, t_2 : \mathbb{N} \to \mathbb{N}$, donde $t_1(n)$ es $o\left(\frac{t_2(n)}{\log t_2(n)}\right)$ y t_2 es construible en tiempo,

$$TIME(t_1(n)) \subsetneq TIME(t_2(n)).$$

Este corolario formaliza la idea de que si una función de tiempo crece suficientemente más rápido que otra, entonces la clase de lenguajes que puede decidir con ese tiempo mayor es estrictamente más amplia. Es una consecuencia directa del teorema de jerarquía temporal, aplicado a dos funciones específicas.

Corollary 4.3 (Corolario 9.13). *[3, p. 343]*

$$\mathbf{P} \subsetneq \mathbf{EXPTIME}.$$

Este corolario es uno de los resultados más importantes derivados de la jerarquía temporal. Como la función 2^n crece mucho más rápido que cualquier polinomio, el teorema implica que existe al menos un lenguaje decidible en tiempo exponencial, pero no en tiempo polinomial.

Así se demuestra que la clase de problemas que pueden resolverse en tiempo polinomial es estrictamente más pequeña que la clase de problemas que pueden resolverse en tiempo exponencial. En otras palabras, existen problemas que son decidibles pero intratables, pues requieren un tiempo exponencial para ser resueltos.

4.1 Preguntas clave

4.1.1 *¿Por qué necesitamos que t(n) sea "construible en tiempo".*

Solución: Porque D necesita calcular t(n) para saber cuánto tiempo puede usar. Si t(n) no pudiera calcularse dentro de t(n) tiempo, la máquina D no podría limitarse correctamente y toda la prueba se caería.

4.1.2 *¿Qué pasa si la máquina M corre en tiempo o(t(n)) pero con una constante tan grande que D no alcanza a simularla completamente?*

Si M efectivamente es "más rápida", pero sus tiempos para entradas pequeñas son enormes, podría parecer que D no puede simularla dentro del límite t(n).

Solución: Usamos entradas del tipo $\langle M \rangle 10^*$ para alargar artificialmente la entrada. Al aumentar la longitud n, la cota $t(n)$ también crece, y eventualmente será suficiente para simular por completo a M en esa entrada alargada. Por eso la diagonalización no falla: siempre existe una entrada lo bastante grande para distinguir A del lenguaje de M.

5 Clases de Complejidad Exponencial y la Completitud de $EQ_{REX\uparrow}$

Véase la sección de Preliminares, donde se introducen las nociones de $EXPSPACE - completitud$ y $EXPTIME - completitud$. De manera semejante a la definición de $EXPSPACE - completitud$ (Definición 2.1:), la noción correspondiente para $EXPTIME - completitud$ (Definición 2.2) es completamente análoga.

Los teoremas jerarquia demuestran que

$$PSPACE \subsetneq EXSPACE$$

$$P \subsetneq EXPTIME$$

Partiendo de esta separación estricta, podemos deducir la intractabilidad de los problemas completos. Si un problema $EXPSPACE - completo$ pudiera resolverse en tiempo polinómico (P), entonces, debido a la propiedad de las reducciones polinómicas, todos los problemas de $EXPSPACE$ podrían resolverse en tiempo polinómico.

Esto implicaría que $EXPSPACE \subseteq P$, causando que la jerarquía colapse y contradiciendo el teorema que establece que EXPSPACE es estrictamente mayor que P. Por lo tanto, es imposible que existan algoritmos eficientes para estos problemas; son demostrablemente intratables.

A continuación se mostrará un problema $EXPSPACE - completo$.

Theorem 5.1. *Sea* $EQ_{REX\uparrow} = \{\langle R_1, R_2 \rangle \mid R_1$ *y* R_2 *son expresiones regulares con exponenciación y* $L(R_1) = L(R_2)\}$. $EQ_{REX\uparrow}$ *es* $EXPSPACE - completo$*[4, p. 4][3, p. 344]*

Proof. (1) $EQ_{REX\uparrow} \in EXPSPACE$

(2) Si $A \in EXPTIME$ entonces $A \leq_p EQ_{REX\uparrow}$
$EQ_{REX\uparrow} \in EXPSPACE$
Sea n la longitud de la entrada $\langle Q, R \rangle$.

(1) Expandimos todas las exponenciaciones en Q y R para obtener expresiones regulares estándar Q' y R'.
(2) Dado que los exponentes están en binario, el valor máximo de un exponente es 2^n. La longitud de las expresiones expandidas es a lo sumo $O(n \cdot 2^n)$.
(3) Convertimos Q' y R' a Autómatas Finitos No Deterministas (NFA) N_Q y N_R. El número de estados de cada autómata es lineal respecto a la longitud de la expresión, es decir, $O(n \cdot 2^n)$.
(4) Verificamos si $L(N_Q) = L(N_R)$. Sabemos que la equivalencia de NFAs se puede decidir en espacio polinómico respecto al tamaño de los autómatas (usando el algoritmo de no-equivalencia y el Teorema de Savitch.
(5) El espacio requerido es $O((\text{tamaño})^2) = O((n \cdot 2^n)^2) = O(2^{2n+2\log n})$, lo cual pertenece a la clase $EXPSPACE$.

Si $A \in EXPTIME$ **entonces** $A \leq_p EQ_{REX\uparrow}$
Sea A un lenguaje arbitrario en EXPSPACE decidido por una Máquina de Turing determinista M que utiliza espacio 2^{n^k} para alguna constante k. Dada una entrada w, construimos en tiempo polinómico dos expresiones regulares R_1 y R_2 tales que:
Construcción:

- $R_1 = \Sigma^*$ (Genera todas las cadenas).
- R_2 generará todas las cadenas que no son historias de computación de rechazo válidas de M sobre w.

Si M acepta w, no existe historia de rechazo, por lo que $L(R_2) = \Sigma^* = L(R_1)$. Si M rechaza, existe una historia válida h, y $L(R_2) = \Sigma^* \setminus \{h\} \neq L(R_1)$.

Construcción Detallada de R_2 Una cadena falla en ser un historial de rechazo si viola al menos una de las condiciones de validez: inicio, transición o finalización.

1. Mal Inicio ($R_{bad\text{-}start}$). Debe generar cualquier cadena que no comience con la configuración inicial $C_1 = q_0 w_1 \dots w_n \sqcup \cdots \sqcup \#$. Sipser descompone esto en la unión de varios errores posibles:
Donde Δ es el alfabeto y Δ_{-x} denota $(\Delta \setminus \{x\})$.

- $S_0 = \Delta_{-q_0}\Sigma^*$: El primer símbolo no es el estado inicial q_0.
- $S_i = \Sigma^i \Delta_{-w_i}\Sigma^*$: El símbolo en la posición $i+1$ no coincide con la entrada w_i (para $1 \leq i \leq n$).
- S_b: Detecta si los símbolos de relleno (blancos) no son correctos. Aquí la exponenciación es conveniente para saltar hasta la zona de blancos, pero en $S_\#$ se vuelve crucial.
- $S_\# = \Sigma^{\uparrow 2^{n^k}} \Delta_{-\#}\Sigma^*$: El símbolo en la posición $2^{n^k}+1$ (donde termina la primera configuración) no es el separador #. Usamos $\Sigma^{\uparrow 2^{n^k}}$ para avanzar exactamente la longitud de una configuración.

2. Mal Movimiento ($R_{bad\text{-}window}$). Esta subexpresión detecta si la configuración C_{t+1} no se sigue válidamente de C_t. En una MT, el contenido de una celda en $t+1$ está determinado localmente por una ventana de 3 celdas en t (izquierda, actual, derecha).

Si en C_t tenemos la secuencia de símbolos abc, y la función de transición de M dicta que el símbolo central debe convertirse en d, entonces hay un error si en C_{t+1} encontramos un símbolo $d' \neq d$ en la posición correspondiente.

La dificultad radica en que, en la cadena lineal del historial, la posición correspondiente en C_{t+1} está separada por exactamente 2^{n^k} símbolos. Sin el operador $\uparrow$, tendríamos que escribir $2^{n^k}-1$ símbolos comodín para conectar abc con el error d'. Esto haría que la expresión regular tuviera longitud exponencial, invalidando la reducción polinómica. Al usar un exponente binario, representamos el número 2^{n^k} con $O(n^k)$ bits, manteniendo el tamaño de R_2 polinómico respecto a n.

3. Mal Final ($R_{bad\text{-}reject}$). Genera cadenas que no contienen el estado de rechazo q_{reject}:

Como R_1 y R_2 se construyen en tiempo polinómico y capturan la aceptación de M, $EQ_{REX\uparrow}$ es $EXPSPACE - completo$. □

i

6 Conclusiones

En este trabajo se ha visto cómo los recursos computacionales, tiempo y espacio, son valiosos. Ofrecer más recursos incrementa el conjunto de problemas que pueden ser resueltos. Los Teoremas de Jerarquía del Espacio y del Tiempo formalizan esta intuición, utilizando la técnica de diagonalización para construir un lenguaje que deliberadamente evita ser resuelto por una máquina con menos recursos.

El Teorema de Jerarquía del Espacio prueba que basta con un poco más de espacio disponible (siempre que sea una cantidad que la máquina pueda medir, conocida como "espacio constructible") para decidir un conjunto estrictamente mayor de lenguajes. En otras palabras, la clase de problemas que se pueden resolver con la función $f_1(n)$ es un subconjunto estricto de la clase que se resuelve con la función $f_2(n)$ si f_2 crece más rápido.

El Teorema de Jerarquía del Tiempo establece una separación análoga. Sin embargo, su demostración es más exigente. Para asegurar que la máquina con más tiempo puede "superar" a la máquina más rápida, la ventaja de tiempo debe ser significativamente mayor, debido al factor logarítmico que surge del costo de simular máquinas paso a paso.

Ambos resultados demuestran de manera irrefutable que las clases de complejidad no colapsan y que la potencia computacional aumenta de manera estricta conforme se incrementan los recursos disponibles.

La consecuencia más significativa de esta jerarquía es la demostración de la intratabilidad. Al probar que P es estrictamente menor que $EXPTIME$, los teoremas nos dicen que existen problemas cuya solución requiere, inherentemente, cantidades enormes de tiempo o espacio (exponencial), haciéndolos impracticables incluso aunque sean teóricamente decidibles. La estructura jerárquica de la complejidad es, por tanto, el fundamento que distingue a los problemas eficientemente resolubles de aquellos que están, demostrablemente, fuera del alcance de la computación práctica.

References

[1] Abhijit Das. 2004. Chapter 4: Hierarchy theorems and intractability. IIT Kharagpur: 17642 Computational Complexity (Notas de clase). https://cse.iitkgp.ac.in/~abhij/course/theory/CC/Spring04/chap4.pdf Recuperado el 23 de noviembre de 2025.

[2] MIT OpenCourseWare. 2021. 21. Hierarchy Theorems. (*Video de YouTube*). https://www.youtube.com/watch?v=vqFRAWeEcUs&list=LL&index=1 Recuperado el 22 de noviembre de 2025.

[3] Michael Sipser. 2006. *Introduction to the Theory of Computation*. Thomson Course Technology, Boston, MA.

[4] Michael Sipser. 2020. Lecture 22: Provably Intractable Problems, Oracles. Diapositivas de MIT OpenCourseWare: 18.404J Theory of Computation. https://ocw.mit.edu/courses/18-404j-theory-of-computation-fall-2020/50cb369d1be3c7fbe0886e318aea13c2_MIT18_404f20_lec22.pdf Recuperado el 22 de noviembre de 2025.