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
NOMBRE DEL AUTOR 3, Otra Universidad/Departamento, Otro País

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 Nombre del Autor 3. 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), 6 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 TEOREMA DEL ESPACIO

Definition 2.1. Una función $f : N \rightarrow N$, 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 $O(f(n))$. [2]

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. [2]. 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,

Authors' addresses: 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; Nombre del Autor 3, Otra Universidad/Departamento, Otra Ciudad, Otro País, autor2@ejemplo.com.

1553-3077/2025/11-ART1 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 2.2. *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 $o(f(n))$. [2]*

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 2.3. *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, $SPACE(f_1(n)) \subsetneq SPACE(f_2(n))$. [2]*

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))}* [1]

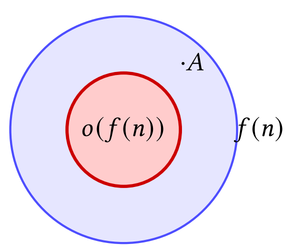

Fig. 1. $SPACE(o(f(n))) \subsetneq SPACE(f(n))$.

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 diferencía 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.

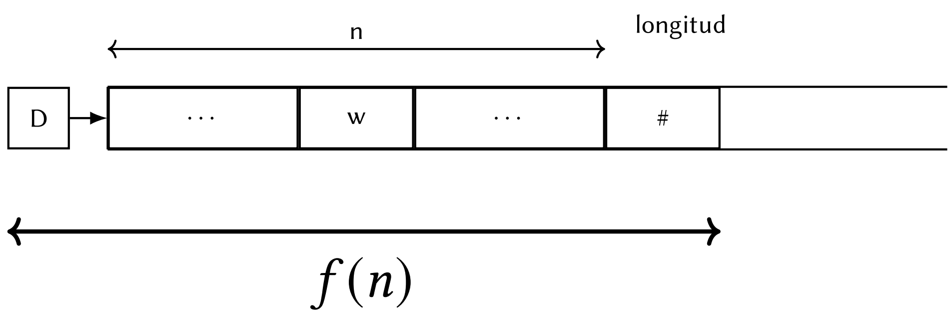

Fig. 2. Punto 1. Solo se permitirá que D use el espacio f(n), de lo contrario tal vez D no este en f(n).

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.

2.1 Problemas

2.1.1 ¿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.$Si w \neq < M > 10^*$ 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.

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

2.1.2 ¿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.

2.1.3 ¿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

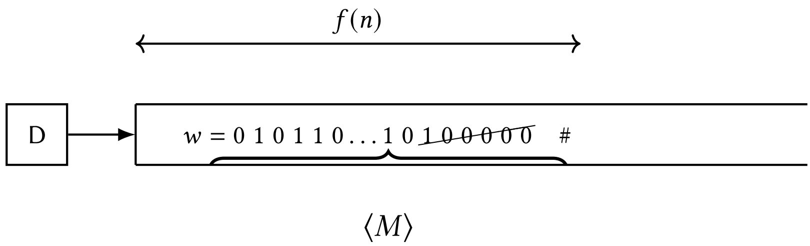

Fig. 3. Problema 2.2.1 <M> abarca desde el primer 0 hasta el último 0, y se debe de tachar desde el último 0 hasta el último 1.

funciones como $log_n, log_n^2, n, n^2, 2^n, ...$ son espacios constructibles.
¿Se puede decir que D tiene como entrada M y simula M sobre sí mismo? Cierto.

3 TEOREMA DEL TIEMPO

Definition 3.1. 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 $o\left(\frac{t(n)}{\log t(n)}\right)$. [2]

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 $o\left(\frac{t(n)}{\log t(n)}\right)$.

Para demostrar el teorema, se define una MT D diseñada para "diferenciarse" de todas las máquinas que corren en tiempo más pequeño que $t(n)/\log t(n)$.[2] La idea es la siguiente:

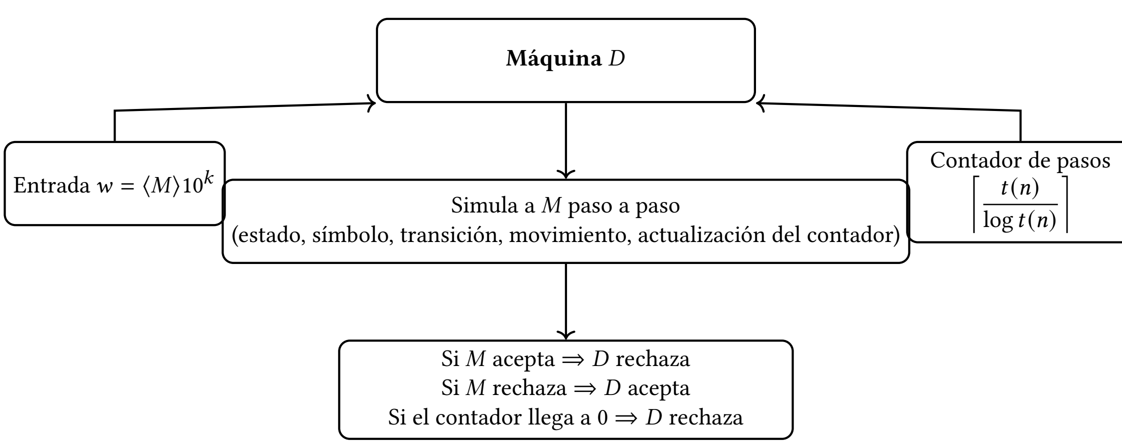

Fig. 4. Esquema de la simulación temporal realizada por la máquina D. La máquina calcula $t(n)$, establece un contador de pasos y simula a M sin exceder el tiempo permitido.

(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;
   - si M rechaza, D acepta.

Esta técnica es un ejemplo clásico de *diagonalización*. La idea de diagonalización utilizada para construir la máquina D también se presenta de manera intuitiva en [1], lo que ayuda a visualizar cómo D garantiza diferenciarse de cualquier máquina que opere por debajo de $t(n)/\log t(n)$.

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 3.2 (Corolario 9.11). *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,*

$$\mathit{TIME}(t_1(n)) \subsetneq \mathit{TIME}(t_2(n)).$$

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 3.3 (Corolario 9.13).

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

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.

3.1 Preguntas clave

3.1.1 ¿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.

3.1.2 ¿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.

4 TEOREMA3

5 CONCLUSIONES

Resumimos nuestros hallazgos aquí.

REFERENCES

[1] 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.

[2] Michael Sipser. 2006. 9.1 Hierarchy Theorems. In *Introduction to the Theory of Computation*. 336.