

UNIVERSIDAD NACIONAL DE INGENIERÍA

Teoría de la Computación

Lo decidible y lo indecidible

Problemas sobre autómatas y gramáticas, la máquina universal y la diagonal

Gerson Garrido

Semana 9

Hoy

Sesión A: lo decidible

1. Las máquinas como entradas
2. Aceptación: A_{DFA}, A_{NFA}, A_{REX}
3. ¿Vacío? E_{DFA}
4. ¿Equivalentes? $\mathrm{EQ}_{\mathrm{DFA}}$
5. Gramáticas: A_{CFG} y E_{CFG}
6. El mapa de los lenguajes

Sesión B: lo indecidible

7. La máquina universal
8. Por qué simular no decide
9. Numerables y la diagonal de Cantor
10. A_{TM} es indecidible
11. El problema de la parada
12. Reconocible y correconocible

Lectura

Apuntes de la semana 9. Apoyo: Sipser, sec. 4.1 y 4.2; Erickson, *Models of Computation*, nota 7. Código en `code/week-09`.

SECCIÓN 01

Las máquinas como entradas

01



De dónde venimos

- Semana 7: una máquina de Turing **acepta**, **rechaza** o **no se detiene**.

De dónde venimos

- Semana 7: una máquina de Turing **acepta**, **rechaza** o **no se detiene**.
- **Decide** un lenguaje: se detiene siempre, y acepta justo sus cadenas.
- **Reconoce** un lenguaje: acepta justo sus cadenas. Con las demás, rechaza o no se detiene.

De dónde venimos

- Semana 7: una máquina de Turing **acepta**, **rechaza** o **no se detiene**.
- **Decide** un lenguaje: se detiene siempre, y acepta justo sus cadenas.
- **Reconoce** un lenguaje: acepta justo sus cadenas. Con las demás, rechaza o no se detiene.
- Tesis de Church-Turing: algoritmo = máquina de Turing. Desde hoy, alto nivel.

De dónde venimos

- Semana 7: una máquina de Turing **acepta**, **rechaza** o **no se detiene**.
- **Decide** un lenguaje: se detiene siempre, y acepta justo sus cadenas.
- **Reconoce** un lenguaje: acepta justo sus cadenas. Con las demás, rechaza o no se detiene.
- Tesis de Church-Turing: algoritmo = máquina de Turing. Desde hoy, alto nivel.

La pregunta de hoy

Semana 1: hay más lenguajes que programas, así que algún problema no tiene algoritmo. **¿Cuál?**

Las máquinas son datos

- La entrada de una máquina es una cadena.
- Todo objeto finito se escribe como cadena: $\langle B \rangle$.
- Un archivo `.dfa` **es** un $\langle B \rangle$.

machines/ends_01.dfa

```
states: q0 q1 q2
alphabet: 0 1
start: q0
accept: q2
delta:
q0 0 q1
q0 1 q0
q1 0 q1
q1 1 q2
q2 0 q1
q2 1 q0
```

Sin las líneas de comentario.

Las máquinas son datos

- La entrada de una máquina es una cadena.
- Todo objeto finito se escribe como cadena: $\langle B \rangle$.
- Un archivo `.dfa` **es** un $\langle B \rangle$.
- Una pregunta sobre autómatas es un **lenguaje de descripciones**:

$$A_{\mathrm{DFA}} = \{ \langle B, w \rangle : B \text{ es un DFA que acepta } w \}$$

machines/ends_01.dfa

```
states: q0 q1 q2
alphabet: 0 1
start: q0
accept: q2
delta:
q0 0 q1
q0 1 q0
q1 0 q1
q1 1 q2
q2 0 q1
q2 1 q0
```

Sin las líneas de comentario.

Las máquinas son datos

- La entrada de una máquina es una cadena.
- Todo objeto finito se escribe como cadena: $\langle B \rangle$.
- Un archivo `.dfa` **es** un $\langle B \rangle$.
- Una pregunta sobre autómatas es un **lenguaje de descripciones**:

$$A_{\mathrm{DFA}} = \{ \langle B, w \rangle : B \text{ es un DFA que acepta } w \}$$

- $\langle B, w \rangle \in A_{\mathrm{DFA}}$ cuando $w \in L(B)$.
- ¿Decidible? ¿Hay **un** algoritmo para todos los DFA?

machines/ends_01.dfa

```
states: q0 q1 q2
alphabet: 0 1
start: q0
accept: q2
delta:
q0 0 q1
q0 1 q0
q1 0 q1
q1 1 q2
q2 0 q1
q2 1 q0
```

Sin las líneas de comentario.

Las preguntas de hoy

Lenguaje	La pregunta	Modelo
A_{DFA}, A_{NFA}, A_{REX}	¿Acepta (o genera) la cadena w?	DFA, NFA, expresión regular
E_{DFA}	¿Su lenguaje es vacío?	DFA
$\mathrm{EQ}_{\mathrm{DFA}}$	¿Reconocen el mismo lenguaje?	Dos DFA
A_{CFG}, E_{CFG}	¿Genera w? ¿Su lenguaje es vacío?	Gramática libre de contexto
A_{TM}, $\mathrm{HALT}_{\mathrm{TM}}$	¿Acepta w? ¿Se detiene con w?	Máquina de Turing

- A de *acceptance*, E de *emptiness*, EQ de *equivalence*. Son los nombres de Sipser.

Las preguntas de hoy

Lenguaje	La pregunta	Modelo
A_{DFA}, A_{NFA}, A_{REX}	¿Acepta (o genera) la cadena w?	DFA, NFA, expresión regular
E_{DFA}	¿Su lenguaje es vacío?	DFA
$\mathrm{EQ}_{\mathrm{DFA}}$	¿Reconocen el mismo lenguaje?	Dos DFA
A_{CFG}, E_{CFG}	¿Genera w? ¿Su lenguaje es vacío?	Gramática libre de contexto
A_{TM}, $\mathrm{HALT}_{\mathrm{TM}}$	¿Acepta w? ¿Se detiene con w?	Máquina de Turing

- A de *acceptance*, E de *emptiness*, EQ de *equivalence*. Son los nombres de Sipser.
- Sesión A: las cuatro primeras filas son decidibles. Sesión B: la última no.

Cómo se demuestra que algo es decidible

M = "Con la entrada $\langle B, w \rangle$, donde B es un DFA y w una cadena:

1. Simular B con la entrada w.
2. Si la simulación termina en un estado de aceptación, *aceptar*. Si no, *rechazar*."

- La primera línea incluye revisar la entrada: si no es un DFA y una cadena, rechaza.

Cómo se demuestra que algo es decidible

M = "Con la entrada $\langle B, w \rangle$, donde B es un DFA y w una cadena:

1. Simular B con la entrada w.
2. Si la simulación termina en un estado de aceptación, *aceptar*. Si no, *rechazar*."

- La primera línea incluye revisar la entrada: si no es un DFA y una cadena, rechaza.
- Hay que demostrar dos cosas: que **termina siempre**, con toda entrada, y que **responde bien**.

Cómo se demuestra que algo es decidible

M = "Con la entrada $\langle B, w \rangle$, donde B es un DFA y w una cadena:

1. Simular B con la entrada w.
2. Si la simulación termina en un estado de aceptación, *aceptar*. Si no, *rechazar*."

- La primera línea incluye revisar la entrada: si no es un DFA y una cadena, rechaza.
- Hay que demostrar dos cosas: que **termina siempre**, con toda entrada, y que **responde bien**.

Lo delicado es que termine

Un algoritmo que responde bien cuando termina, pero a veces no termina, **reconoce** y no decide.

SECCIÓN 02

Aceptación

02

A_{DFA} es decidible: decidir es simular

La máquina M de la diapositiva anterior.

terminal

```
$ ../build/a_dfa ends_01.dfa 0101 ''
B = ends_01.dfa, 3 estados
w = 0101
  q0 -0-> q1 -1-> q2 -0-> q1 -1-> q2
  4 pasos; termina en q2, que está en F: ⟨B, w⟩ ∈ A_DFA
w = ε
  q0
  0 pasos; termina en q0, que no está en F: ⟨B, w⟩ ∉ A_DFA
```

Es el laboratorio de la semana 1. Hoy los comandos se corren desde `code/week-09/machines`.

A_{DFA} es decidible: decidir es simular

La máquina M de la diapositiva anterior.

- **Termina:** un paso por símbolo. Son $|w|$ búsquedas en la tabla de δ.

terminal

```
$ ../build/a_dfa ends_01.dfa 0101 ''
B = ends_01.dfa, 3 estados
w = 0101
  q0 -0-> q1 -1-> q2 -0-> q1 -1-> q2
  4 pasos; termina en q2, que está en F: ⟨B, w⟩ ∈ A_DFA
w = ε
  q0
  0 pasos; termina en q0, que no está en F: ⟨B, w⟩ ∉ A_DFA
```

Es el laboratorio de la semana 1. Hoy los comandos se corren desde `code/week-09/machines`.

A_{DFA} es decidible: decidir es simular

La máquina M de la diapositiva anterior.

- **Termina:** un paso por símbolo. Son $|w|$ búsquedas en la tabla de δ.
- **Responde bien:** al final tiene $\delta^*(q_0, w)$, y mira si está en F.

terminal

```
$ ../build/a_dfa ends_01.dfa 0101 ''
B = ends_01.dfa, 3 estados
w = 0101
  q0 -0-> q1 -1-> q2 -0-> q1 -1-> q2
  4 pasos; termina en q2, que está en F: ⟨B, w⟩ ∈ A_DFA
w = ε
  q0
  0 pasos; termina en q0, que no está en F: ⟨B, w⟩ ∉ A_DFA
```

Es el laboratorio de la semana 1. Hoy los comandos se corren desde `code/week-09/machines`.

A_{DFA} es decidible: decidir es simular

La máquina M de la diapositiva anterior.

- **Termina:** un paso por símbolo. Son $|w|$ búsquedas en la tabla de δ.
- **Responde bien:** al final tiene $\delta^*(q_0, w)$, y mira si está en F.

Lo que sostiene todo

Un DFA **siempre termina**. La simulación lo hereda.

terminal

```
$ ../build/a_dfa ends_01.dfa 0101 ''
B = ends_01.dfa, 3 estados
w = 0101
  q0 -0-> q1 -1-> q2 -0-> q1 -1-> q2
  4 pasos; termina en q2, que está en F: ⟨B, w⟩ ∈ A_DFA
w = ε
  q0
  0 pasos; termina en q0, que no está en F: ⟨B, w⟩ ∉ A_DFA
```

Es el laboratorio de la semana 1. Hoy los comandos se corren desde `code/week-09/machines`.

A_{NFA} y A_{REX}: convertir y simular



N' = "Con la entrada $\langle N, w \rangle$, donde N es un NFA y w una cadena:

1. Convertir N en un DFA equivalente D, con la construcción de subconjuntos.
2. Correr M con la entrada $\langle D, w \rangle$. Si M acepta, *aceptar*. Si rechaza, *rechazar*."

A_{NFA} y A_{REX}: convertir y simular



N' = "Con la entrada $\langle N, w \rangle$, donde N es un NFA y w una cadena:

1. Convertir N en un DFA equivalente D, con la construcción de subconjuntos.
2. Correr M con la entrada $\langle D, w \rangle$. Si M acepta, *aceptar*. Si rechaza, *rechazar*."

- Termina: a lo sumo $2^{|Q|}$ subconjuntos, y M decide. Correcta: $L(D) = L(N)$, semana 2.

A_{NFA} y A_{REX}: convertir y simular



N' = "Con la entrada $\langle N, w \rangle$, donde N es un NFA y w una cadena:

1. Convertir N en un DFA equivalente D, con la construcción de subconjuntos.
2. Correr M con la entrada $\langle D, w \rangle$. Si M acepta, *aceptar*. Si rechaza, *rechazar*."

- Termina: a lo sumo $2^{|Q|}$ subconjuntos, y M decide. Correcta: $L(D) = L(N)$, semana 2.
- Una máquina puede llamar a otra, **si la otra termina**.
- Las demostraciones por construcción de antes ahora son pasos de un algoritmo.

SECCIÓN 03

¿Vacío? ¿Equivalentes?

03



E_{DFA}: un intento que no decide

$$E_{\mathrm{DFA}} = \{\, \langle A \rangle : A \text{ es un DFA y } L(A) = \emptyset \,\}$$

Probar cadenas

"Con la entrada $\langle A \rangle$: para cada cadena w de Σ^*, de las más cortas a las más largas, simular A con w. Si A acepta, *rechazar*."

E_{DFA}: un intento que no decide

$$E_{\mathrm{DFA}} = \{\, \langle A \rangle : A \text{ es un DFA y } L(A) = \emptyset \,\}$$

Probar cadenas

"Con la entrada $\langle A \rangle$: para cada cadena w de Σ^*, de las más cortas a las más largas, simular A con w. Si A acepta, *rechazar*."

- Si $L(A) \neq \emptyset$: encuentra una cadena y rechaza. Bien.

E_{DFA}: un intento que no decide

$$E_{\mathrm{DFA}} = \{\, \langle A \rangle : A \text{ es un DFA y } L(A) = \emptyset \,\}$$

Probar cadenas

"Con la entrada $\langle A \rangle$: para cada cadena w de Σ^*, de las más cortas a las más largas, simular A con w. Si A acepta, *rechazar*."

- Si $L(A) \neq \emptyset$: encuentra una cadena y rechaza. Bien.
- Si $L(A) = \emptyset$: sigue probando **para siempre**. Nunca acepta.

E_{DFA}: un intento que no decide

$$E_{\mathrm{DFA}} = \{\, \langle A \rangle : A \text{ es un DFA y } L(A) = \emptyset \,\}$$

Probar cadenas

"Con la entrada $\langle A \rangle$: para cada cadena w de Σ^*, de las más cortas a las más largas, simular A con w. Si A acepta, *rechazar*."

- Si $L(A) \neq \emptyset$: encuentra una cadena y rechaza. Bien.
- Si $L(A) = \emptyset$: sigue probando **para siempre**. Nunca acepta.

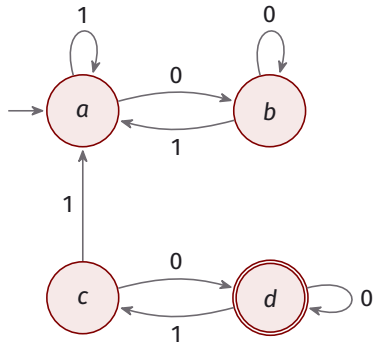
La salida

No mirar las cadenas, que son infinitas. Mirar el **autómata**, que es finito: ¿hay un camino del estado inicial a uno de aceptación?

E_{DFA} es decidible: marcar lo alcanzable

T = "Con la entrada $\langle A \rangle$, donde A es un DFA:

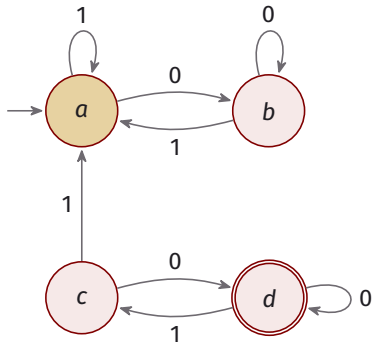
1. Marcar el estado inicial.
2. Repetir hasta no marcar nada nuevo: marcar todo estado al que llega una transición desde uno marcado.
3. Si ningún estado de aceptación quedó marcado, *aceptar*. Si no, *rechazar*."



E_{DFA} es decidible: marcar lo alcanzable

T = "Con la entrada $\langle A \rangle$, donde A es un DFA:

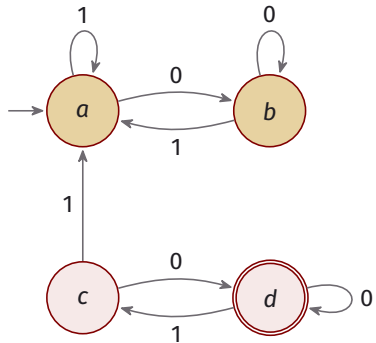
1. Marcar el estado inicial.
2. Repetir hasta no marcar nada nuevo: marcar todo estado al que llega una transición desde uno marcado.
3. Si ningún estado de aceptación quedó marcado, *aceptar*. Si no, *rechazar*."



E_{DFA} es decidible: marcar lo alcanzable

T = "Con la entrada $\langle A \rangle$, donde A es un DFA:

1. Marcar el estado inicial.
2. Repetir hasta no marcar nada nuevo: marcar todo estado al que llega una transición desde uno marcado.
3. Si ningún estado de aceptación quedó marcado, *aceptar*. Si no, *rechazar*."



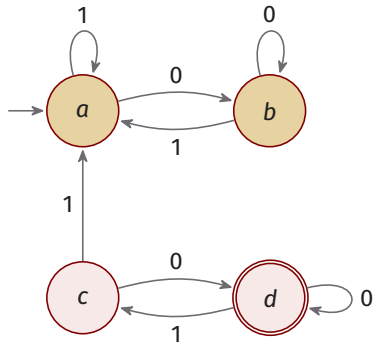
Marcados: a, b. $F = \{d\}$ y aun así $L(A) = \emptyset$.

E_{DFA} es decidible: marcar lo alcanzable

T = "Con la entrada $\langle A \rangle$, donde A es un DFA:

1. Marcar el estado inicial.
2. Repetir hasta no marcar nada nuevo: marcar todo estado al que llega una transición desde uno marcado.
3. Si ningún estado de aceptación quedó marcado, *aceptar*. Si no, *rechazar*."

- **Termina:** cada vuelta marca algo o es la última. A lo sumo $|Q|$ vueltas.



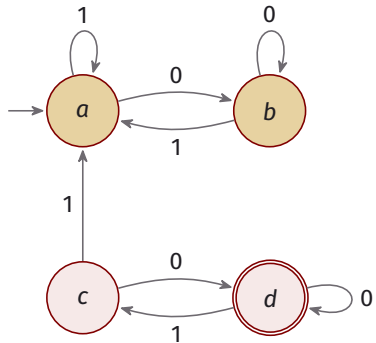
Marcados: a, b. $F = \{d\}$ y aun así $L(A) = \emptyset$.

E_{DFA} es decidible: marcar lo alcanzable

T = "Con la entrada $\langle A \rangle$, donde A es un DFA:

1. Marcar el estado inicial.
2. Repetir hasta no marcar nada nuevo: marcar todo estado al que llega una transición desde uno marcado.
3. Si ningún estado de aceptación quedó marcado, *aceptar*. Si no, *rechazar*."

- **Termina:** cada vuelta marca algo o es la última. A lo sumo $|Q|$ vueltas.
- **Correcta:** marcado $\iff$ alcanzable.



Marcados: a, b. $F = \{d\}$ y aun así $L(A) = \emptyset$.

E_{DFA}: la demostración y el programa

Marcado $\iff$ alcanzable

- $\Rightarrow$: inducción sobre el orden de las marcas. Si $r = \delta(q, a)$ y $q = \delta^*(q_0, x)$, entonces $r = \delta^*(q_0, xa)$.

terminal

```
$ ../build/e_dfa unreachable_accept.dfa
A = unreachable_accept.dfa, 4 estados, F = {d}
ronda 0: se marca a
ronda 1: se marca b
marcados: 2 de 4
ningún estado de aceptación está marcado: L(A) = ∅, ⟨A⟩ ∈ E_DFA
```

E_{DFA}: la demostración y el programa

Marcado $\iff$ alcanzable

- $\Rightarrow$: inducción sobre el orden de las marcas. Si $r = \delta(q, a)$ y $q = \delta^*(q_0, x)$, entonces $r = \delta^*(q_0, xa)$.
- $\Leftarrow$: inducción sobre $|w|$. Si $\delta^*(q_0, x)$ está marcado y su destino con a no, el ciclo no habría terminado.

terminal

```
$ ../build/e_dfa unreachable_accept.dfa
A = unreachable_accept.dfa, 4 estados, F = {d}
ronda 0: se marca a
ronda 1: se marca b
marcados: 2 de 4
ningún estado de aceptación está marcado: L(A) = ∅, ⟨A⟩ ∈ E_DFA
```

E_{DFA}: la demostración y el programa

Marcado $\iff$ alcanzable

- $\Rightarrow$: inducción sobre el orden de las marcas. Si $r = \delta(q, a)$ y $q = \delta^*(q_0, x)$, entonces $r = \delta^*(q_0, xa)$.
- $\Leftarrow$: inducción sobre $|w|$. Si $\delta^*(q_0, x)$ está marcado y su destino con a no, el ciclo no habría terminado.
- $L(A) \neq \emptyset \iff$ algún estado de F es alcanzable.

terminal

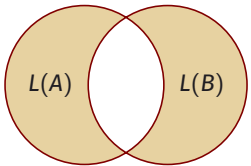
```
$ ../build/e_dfa unreachable_accept.dfa
A = unreachable_accept.dfa, 4 estados, F = {d}
ronda 0: se marca a
ronda 1: se marca b
marcados: 2 de 4
ningún estado de aceptación está marcado: L(A) = ∅, ⟨A⟩ ∈ E_DFA
```

Error común

$L(A) = \emptyset$ no quiere decir $F = \emptyset$.

$\mathrm{EQ}_{\mathrm{DFA}}$: convertir la pregunta en otra

$$\mathrm{EQ}_{\mathrm{DFA}} = \{\, \langle A, B \rangle : A \text{ y } B \text{ son DFA y } L(A) = L(B) \,\}$$

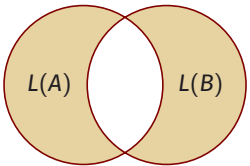


La **diferencia simétrica**: las cadenas que están en exactamente uno.

- No sirve comparar diagramas ni probar cadenas.

$\mathrm{EQ}_{\mathrm{DFA}}$: convertir la pregunta en otra

$$\mathrm{EQ}_{\mathrm{DFA}} = \{\, \langle A, B \rangle : A \text{ y } B \text{ son DFA y } L(A) = L(B) \,\}$$

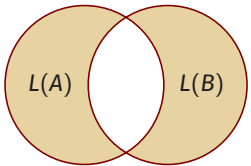


La **diferencia simétrica**: las cadenas que están en exactamente uno.

- No sirve comparar diagramas ni probar cadenas.
- $L(A) \triangle L(B) = \big(L(A) \cap \overline{L(B)}\big) \cup \big(\overline{L(A)} \cap L(B)\big)$

$\mathrm{EQ}_{\mathrm{DFA}}$: convertir la pregunta en otra

$$\mathrm{EQ}_{\mathrm{DFA}} = \{\, \langle A, B \rangle : A \text{ y } B \text{ son DFA y } L(A) = L(B) \,\}$$

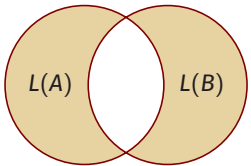


La **diferencia simétrica**: las cadenas que están en exactamente uno.

- No sirve comparar diagramas ni probar cadenas.
- $L(A) \triangle L(B) = \big(L(A) \cap \overline{L(B)}\big) \cup \big(\overline{L(A)} \cap L(B)\big)$
- $L(A) = L(B)$ si y solo si $L(A) \triangle L(B) = \emptyset$.

$\mathrm{EQ}_{\mathrm{DFA}}$: convertir la pregunta en otra

$$\mathrm{EQ}_{\mathrm{DFA}} = \{ \langle A, B \rangle : A \text{ y } B \text{ son DFA y } L(A) = L(B) \}$$



La **diferencia simétrica**: las cadenas que están en exactamente uno.

- No sirve comparar diagramas ni probar cadenas.
- $L(A) \triangle L(B) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right)$
- $L(A) = L(B)$ si y solo si $L(A) \triangle L(B) = \emptyset$.
- Y "¿es vacío?" ya lo sabemos decidir, si hay un DFA para ese lenguaje.
- Lo hay: los regulares son cerrados bajo complemento, intersección y unión (semana 1).

El autómata de la diferencia simétrica

El DFA C, con la construcción del producto

Estados: los pares (p, q). Inicial: (q_A, q_B). $\delta_C((p, q), a) = (\delta_A(p, a), \delta_B(q, a))$

$F_C = \{\, (p, q) : \textbf{exactamente uno}$ de $p \in F_A,\ q \in F_B \,\}$

El autómata de la diferencia simétrica

El DFA C, con la construcción del producto

Estados: los pares (p, q). Inicial: (q_A, q_B). $\delta_C((p,q),a) = (\delta_A(p,a), \delta_B(q,a))$

$F_C = \{\, (p,q) : \textbf{exactamente uno}$ de $p \in F_A,\ q \in F_B \,\}$

- Lema del producto (semana 1): $\delta_C^*((p,q),w) = (\delta_A^*(p,w), \delta_B^*(q,w))$.
- C acepta $w \iff$ exactamente uno de A, B acepta w: $L(C) = L(A) \triangle L(B)$.

El autómata de la diferencia simétrica

El DFA C, con la construcción del producto

Estados: los pares (p, q). Inicial: (q_A, q_B). $\delta_C((p,q),a) = (\delta_A(p,a), \delta_B(q,a))$

$F_C = \{\, (p,q) : \textbf{exactamente uno}$ de $p \in F_A,\ q \in F_B \,\}$

- Lema del producto (semana 1): $\delta_C^*((p,q),w) = (\delta_A^*(p,w), \delta_B^*(q,w))$.
- C acepta $w \iff$ exactamente uno de A, B acepta w: $L(C) = L(A) \triangle L(B)$.

F = "Con la entrada $\langle A, B\rangle$, donde A y B son DFA:

1. Construir C.
2. Correr T con la entrada $\langle C\rangle$. Si T acepta, *aceptar*. Si rechaza, *rechazar*."

El autómata de la diferencia simétrica

El DFA C, con la construcción del producto

Estados: los pares (p, q). Inicial: (q_A, q_B). $\delta_C((p,q),a) = (\delta_A(p,a), \delta_B(q,a))$

$F_C = \{\, (p,q) : \textbf{exactamente uno}$ de $p \in F_A,\ q \in F_B \,\}$

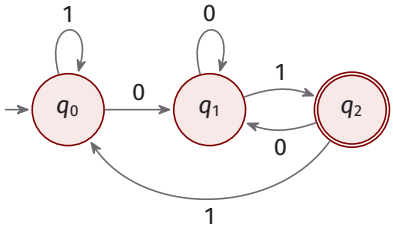
- Lema del producto (semana 1): $\delta_C^*((p,q),w) = (\delta_A^*(p,w), \delta_B^*(q,w))$.
- C acepta $w \iff$ exactamente uno de A, B acepta w: $L(C) = L(A) \triangle L(B)$.

F = "Con la entrada $\langle A, B\rangle$, donde A y B son DFA:

1. Construir C.
2. Correr T con la entrada $\langle C\rangle$. Si T acepta, *aceptar*. Si rechaza, *rechazar*."

Termina: C tiene $|Q_A| \cdot |Q_B|$ estados y T decide. Correcta: $L(C) = \emptyset \iff L(A) = L(B)$.

Dos autómatas para “termina en 01”

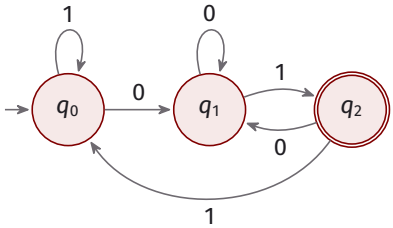


- A: 3 estados, el de la semana 1.
- B: 7 estados. Recuerda los dos últimos símbolos: e, s_0, s_1, s_{00}, s_{01}, s_{10}, s_{11}.

terminal

```
$ ../build/eq_dfa ends_01.dfa ends_01_history.dfa --producto
A = ends_01.dfa, 3 estados
B = ends_01_history.dfa, 7 estados
C, el producto completo: 21 estados, 8 de aceptación, 7 alcanzables
L(C) = ∅
equivalentes: L(A) = L(B), ⟨A, B⟩ ∈ EQ_DFA
```

Dos autómatas para “termina en 01”

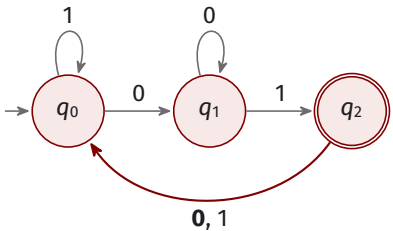


- A: 3 estados, el de la semana 1.
- B: 7 estados. Recuerda los dos últimos símbolos: e, s_0, s_1, s_{00}, s_{01}, s_{10}, s_{11}.
- $3 \cdot 7 = 21$ pares, $6 + 2 = 8$ de aceptación, ninguno alcanzable.

terminal

```
$ ../build/eq_dfa ends_01.dfa ends_01_history.dfa --producto
A = ends_01.dfa, 3 estados
B = ends_01_history.dfa, 7 estados
C, el producto completo: 21 estados, 8 de aceptación, 7 alcanzables
L(C) = ∅
equivalentes: L(A) = L(B), ⟨A, B⟩ ∈ EQ_DFA
```

Un autómata equivocado



B: desde q_2, con 0, vuelve a q_0 y olvida ese 0.

Con las 15 cadenas de longitud ≤ 3, *A* y *B* responden igual.

```
terminal

$ ../build/eq_dfa ends_01.dfa ends_01_wrong.dfa --pares
A = ends_01.dfa, 3 estados
B = ends_01_wrong.dfa, 3 estados
pares visitados: 5 de 9
  (q0,q0)
  (q1,q1)
  (q2,q2)
  (q1,q0)
  (q2,q0)  <- uno acepta y el otro no
distintos: ⟨A, B⟩ ∉ EQ_DFA
```

La última línea de la salida: `una cadena más corta que los separa: 0101 (4 símbolos); A la acepta y B la rechaza`

Sin construir el producto

- F construye $|Q_A| \cdot |Q_B|$ pares y T visita pocos. Con 50 000 estados cada uno: 2 500 000 000.

terminal

```
$ ../build/eq_dfa ends_01.dfa ends_01_history.dfa --pares
A = ends_01.dfa, 3 estados
B = ends_01_history.dfa, 7 estados
pares visitados: 7 de 21
  (q0,e)
  (q1,s0)
  (q0,s1)
  (q1,s00)
  (q2,s01)
  (q1,s10)
  (q0,s11)
equivalentes: L(A) = L(B), ⟨A, B⟩ ∈ EQ_DFA
```

Sin construir el producto

- F construye $|Q_A| \cdot |Q_B|$ pares y T visita pocos. Con 50 000 estados cada uno: 2 500 000 000.
- Las transiciones de C se calculan **en el momento**.

Búsqueda en anchura sobre los pares

Desde (q_A, q_B). Si en el par que sale de la cola exactamente uno acepta: **distintos**. Si no, encolar sus sucesores nuevos. Cola vacía: **equivalentes**.

terminal

```
$ ../build/eq_dfa ends_01.dfa ends_01_history.dfa --pares
A = ends_01.dfa, 3 estados
B = ends_01_history.dfa, 7 estados
pares visitados: 7 de 21
  (q0,e)
  (q1,s0)
  (q0,s1)
  (q1,s00)
  (q2,s01)
  (q1,s10)
  (q0,s11)
equivalentes: L(A) = L(B), ⟨A, B⟩ ∈ EQ_DFA
```

Sin construir el producto

- F construye $|Q_A| \cdot |Q_B|$ pares y T visita pocos. Con 50 000 estados cada uno: 2 500 000 000.
- Las transiciones de C se calculan **en el momento**.

Búsqueda en anchura sobre los pares

Desde (q_A, q_B). Si en el par que sale de la cola exactamente uno acepta: **distintos**. Si no, encolar sus sucesores nuevos. Cola vacía: **equivalentes**.

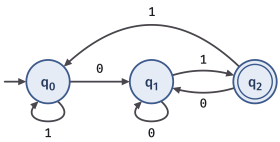
terminal

```
$ ../build/eq_dfa ends_01.dfa ends_01_history.dfa --pares
A = ends_01.dfa, 3 estados
B = ends_01_history.dfa, 7 estados
pares visitados: 7 de 21
  (q0,e)
  (q1,s0)
  (q0,s1)
  (q1,s00)
  (q2,s01)
  (q1,s10)
  (q0,s11)
equivalentes: L(A) = L(B), ⟨A, B⟩ ∈ EQ_DFA
```

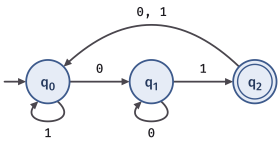
Cada estado de B aparece con un solo estado de A: el que significa lo mismo. Es el laboratorio.

La búsqueda, par por par

A = ends_01.dfa



B = ends_01_wrong.dfa



el error: desde q_2, con 0, vuelve a q_0

Los pares (estado de A, estado de B), en el orden en que salen de la cola



cola: (q_0, q_0)

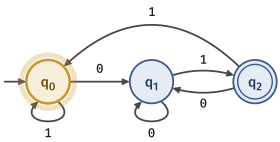
marcados: 1 de 9

Se marca el par inicial, (q_0, q_0), y entra a la cola.

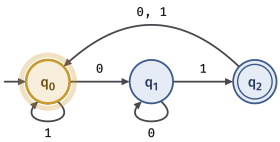
Las transiciones de un par se calculan en el momento: δ_A en A y δ_B en B.

La búsqueda, par por par

A = ends_01.dfa

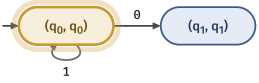


B = ends_01_wrong.dfa



el error: desde q_2, con 0, vuelve a q_0

Los pares (estado de A, estado de B), en el orden en que salen de la cola



cola: (q_1, q_1)

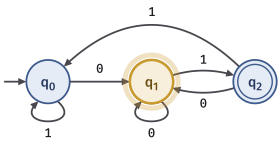
marcados: 2 de 9

Sale (q_0, q_0) de la cola: ninguno acepta.

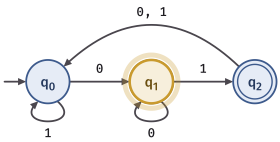
Con 0 va a (q_1, q_1), que es nuevo; con 1 va a (q_0, q_0), ya marcado.

La búsqueda, par por par

A = ends_01.dfa

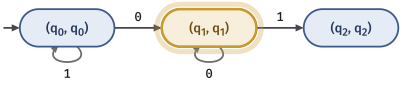


B = ends_01_wrong.dfa



el error: desde q_2, con 0, vuelve a q_0

Los pares (estado de A, estado de B), en el orden en que salen de la cola



cola: (q_2, q_2)

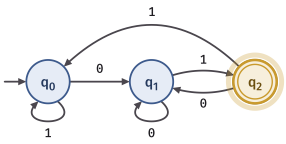
marcados: 3 de 9

Sale (q_1, q_1) de la cola: ninguno acepta.

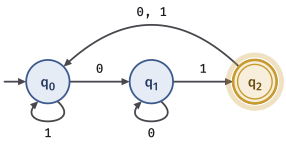
Con 0 va a (q_1, q_1), ya marcado; con 1 va a (q_2, q_2), que es nuevo.

La búsqueda, par por par

A = ends_01.dfa

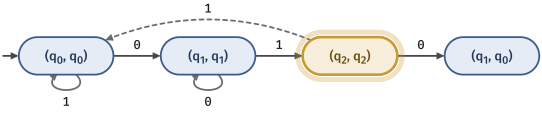


B = ends_01_wrong.dfa



el error: desde q_2, con 0, vuelve a q_0

Los pares (estado de A, estado de B), en el orden en que salen de la cola



cola: (q_1, q_0)

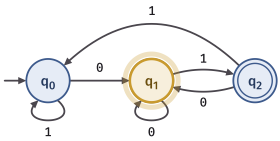
marcados: 4 de 9

Sale (q_2, q_2) de la cola: los dos aceptan.

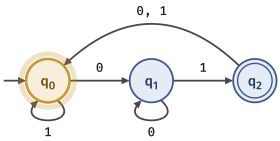
Con 0 va a (q_1, q_0), que es nuevo; con 1 va a (q_0, q_0), ya marcado.

La búsqueda, par por par

A = ends_01.dfa

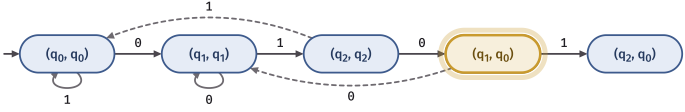


B = ends_01_wrong.dfa



el error: desde q_2, con 0, vuelve a q_0

Los pares (estado de A, estado de B), en el orden en que salen de la cola



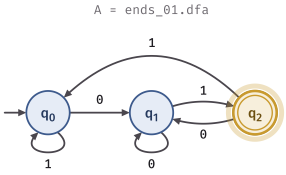
cola: (q_2, q_0)

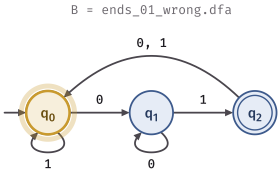
marcados: 5 de 9

Sale (q_1, q_0) de la cola: ninguno acepta.

Con 0 va a (q_1, q_1), ya marcado; con 1 va a (q_2, q_0), que es nuevo.

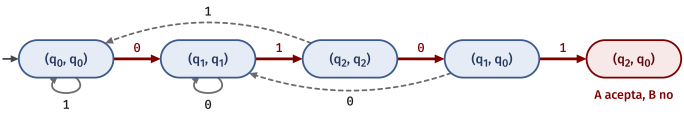
La búsqueda, par por par





el error: desde q_2, con 0, vuelve a q_0

Los pares (estado de A, estado de B), en el orden en que salen de la cola



cola: vacía

marcados: 5 de 9

Sale (q_2, q_0) de la cola: A acepta y B no. La búsqueda termina: distintos.

Las flechas que llevaron hasta ese par dan la cadena `0101`, una de las más cortas que los separan.

La cadena que separa

- **Es de las más cortas.** La búsqueda en anchura saca los pares por distancia creciente, y la distancia de un par es la longitud de la cadena más corta que llega a él.

La cadena que separa

- **Es de las más cortas.** La búsqueda en anchura saca los pares por distancia creciente, y la distancia de un par es la longitud de la cadena más corta que llega a él.
- **Nunca es muy larga.** Si A y B son distintos, alguna cadena de longitud menor que $|Q_A| \cdot |Q_B|$ los separa.
  - Una cadena mínima w de $L(C)$ pasa por $|w| + 1$ pares.
  - Si se repitiera un par, se quita el tramo entre las dos visitas: una más corta.

La cadena que separa

- **Es de las más cortas.** La búsqueda en anchura saca los pares por distancia creciente, y la distancia de un par es la longitud de la cadena más corta que llega a él.
- **Nunca es muy larga.** Si A y B son distintos, alguna cadena de longitud menor que $|Q_A| \cdot |Q_B|$ los separa.
  - Una cadena mínima w de $L(C)$ pasa por $|w| + 1$ pares.
  - Si se repitiera un par, se quita el tramo entre las dos visitas: una más corta.
- Eso da un tercer algoritmo: probar todas las cadenas hasta esa longitud. Termina, y es exponencial.

La cadena que separa

- **Es de las más cortas.** La búsqueda en anchura saca los pares por distancia creciente, y la distancia de un par es la longitud de la cadena más corta que llega a él.
- **Nunca es muy larga.** Si A y B son distintos, alguna cadena de longitud menor que $|Q_A| \cdot |Q_B|$ los separa.
  - Una cadena mínima w de $L(C)$ pasa por $|w| + 1$ pares.
  - Si se repitiera un par, se quita el tramo entre las dos visitas: una más corta.
- Eso da un tercer algoritmo: probar todas las cadenas hasta esa longitud. Termina, y es exponencial.

Idea clave

Una pregunta sobre dos autómatas se convirtió en una pregunta, ya resuelta, sobre uno. Convertir un problema en otro es la técnica de la semana 10.

SECCIÓN 04

Gramáticas

04

A_{CFG}: cuándo dejar de buscar

$$A_{\mathrm{CFG}} = \{\, \langle G, w \rangle : G \text{ es una gramática que genera } w \,\}$$

Probar derivaciones

"Con la entrada $\langle G, w \rangle$: listar las derivaciones de G, primero las de 1 paso, después las de 2, y así. Si alguna termina en w, *aceptar*."

A_{CFG}: cuándo dejar de buscar

$$A_{\mathrm{CFG}} = \{\, \langle G, w \rangle : G \text{ es una gramática que genera } w \,\}$$

Probar derivaciones

"Con la entrada $\langle G, w \rangle$: listar las derivaciones de G, primero las de 1 paso, después las de 2, y así. Si alguna termina en w, *aceptar*."

- Si G genera w: la encuentra.

A_{CFG}: cuándo dejar de buscar

$$A_{\mathrm{CFG}} = \{\, \langle G, w \rangle : G \text{ es una gramática que genera } w \,\}$$

Probar derivaciones

"Con la entrada $\langle G, w \rangle$: listar las derivaciones de G, primero las de 1 paso, después las de 2, y así. Si alguna termina en w, *aceptar*."

- Si G genera w: la encuentra.
- Si no: con una regla como $S \to SS$ hay derivaciones de cualquier longitud. **No termina.**

A_{CFG}: cuándo dejar de buscar

$$A_{\mathrm{CFG}} = \{\, \langle G, w \rangle : G \text{ es una gramática que genera } w \,\}$$

Probar derivaciones

"Con la entrada $\langle G, w \rangle$: listar las derivaciones de G, primero las de 1 paso, después las de 2, y así. Si alguna termina en w, *aceptar*."

- Si G genera w: la encuentra.
- Si no: con una regla como $S \to SS$ hay derivaciones de cualquier longitud. **No termina.**
- Falta una **cota** para los pasos de una derivación de w. La da la forma normal de Chomsky: solo reglas $A \to BC$ y $A \to a$ (y $S \to \varepsilon$), y ningún paso se da sin avanzar.

Exactamente $2n - 1$ pasos

Lema

Si G está en forma normal de Chomsky y $|w| = n \geq 1$, toda derivación de w tiene exactamente $2n - 1$ pasos.

Exactamente $2n - 1$ pasos

Lema

Si G está en forma normal de Chomsky y $|w| = n \geq 1$, toda derivación de w tiene exactamente $2n - 1$ pasos.

- Un paso con $A \to BC$ alarga la cadena en 1. De longitud 1 a longitud n: $n - 1$ pasos.

Exactamente $2n - 1$ pasos

Lema

Si G está en forma normal de Chomsky y $|w| = n \geq 1$, toda derivación de w tiene exactamente $2n - 1$ pasos.

- Un paso con $A \to BC$ alarga la cadena en 1. De longitud 1 a longitud n: $n - 1$ pasos.
- Un paso con $A \to a$ pone un terminal. Hay n terminales: n pasos.
- Total: $(n - 1) + n = 2n - 1$.

Exactamente $2n - 1$ pasos

Lema

Si G está en forma normal de Chomsky y $|w| = n \geq 1$, toda derivación de w tiene exactamente $2n - 1$ pasos.

- Un paso con $A \to BC$ alarga la cadena en 1. De longitud 1 a longitud n: $n - 1$ pasos.
- Un paso con $A \to a$ pone un terminal. Hay n terminales: n pasos.
- Total: $(n - 1) + n = 2n - 1$.

$$S_0 \Rightarrow AT \Rightarrow \mathtt{a}T \Rightarrow \mathtt{a}SB \Rightarrow \mathtt{a}ABB \Rightarrow \mathtt{aa}BB \Rightarrow \mathtt{aab}B \Rightarrow \mathtt{aabb}$$

$n = 4$: 7 pasos, 3 con una regla $A \to BC$ y 4 con una regla $A \to a$. Gramática: `machines/anbn_chomsky.cfg`.

A_{CFG} es decidible

S = "Con la entrada $\langle G, w \rangle$, donde G es una gramática y w una cadena:

1. Convertir G a forma normal de Chomsky.
2. Listar todas las derivaciones de $2n - 1$ pasos, con $n = |w|$. Si $n = 0$, las de 1 paso.
3. Si alguna genera w, *aceptar*. Si ninguna, *rechazar*."

terminal

```
$ ../build/a_cfg anbn_chomsky.cfg aabb abab ''
G = anbn_chomsky.cfg, 8 reglas, en forma normal de Chomsky: sí
w = aabb: n = 4, derivaciones de 7 pasos; G la genera, ⟨G, w⟩ ∈ A_CFG
w = abab: n = 4, derivaciones de 7 pasos; G no la genera, ⟨G, w⟩ ∉ A_CFG
w = ε: n = 0, derivaciones de 1 paso; G la genera, ⟨G, w⟩ ∈ A_CFG
```

A_{CFG} es decidible

S = "Con la entrada $\langle G, w\rangle$, donde G es una gramática y w una cadena:

1. Convertir G a forma normal de Chomsky.
2. Listar todas las derivaciones de $2n - 1$ pasos, con $n = |w|$. Si $n = 0$, las de 1 paso.
3. Si alguna genera w, *aceptar*. Si ninguna, *rechazar*."

terminal

```
$ ../build/a_cfg anbn_chomsky.cfg aabb abab ''
G = anbn_chomsky.cfg, 8 reglas, en forma normal de Chomsky: sí
w = aabb: n = 4, derivaciones de 7 pasos; G la genera, ⟨G, w⟩ ∈ A_CFG
w = abab: n = 4, derivaciones de 7 pasos; G no la genera, ⟨G, w⟩ ∉ A_CFG
w = ε: n = 0, derivaciones de 1 paso; G la genera, ⟨G, w⟩ ∈ A_CFG
```

Termina, pero es exponencial en n. **CYK** (semana 5) decide lo mismo con una tabla finita, en tiempo del orden de $n^3 \cdot |G|$.

E_{CFG}: marcar desde abajo

R = "Con la entrada $\langle G \rangle$, donde G es una gramática:

1. Marcar todos los terminales.
2. Repetir hasta no marcar nada nuevo: marcar toda variable A con una regla $A \to U_1 \cdots U_k$ cuyos símbolos estén todos marcados.
3. Si la variable inicial no quedó marcada, *aceptar*. Si no, *rechazar*."

- En un DFA se marca desde el inicio **hacia adelante**.
- En una gramática, desde los terminales **hacia arriba**.
- $A \to \varepsilon$: $k = 0$, se marca enseguida.

E_{CFG}: marcar desde abajo

R = "Con la entrada $\langle G \rangle$, donde G es una gramática:

1. Marcar todos los terminales.
2. Repetir hasta no marcar nada nuevo: marcar toda variable A con una regla $A \to U_1 \cdots U_k$ cuyos símbolos estén todos marcados.
3. Si la variable inicial no quedó marcada, *aceptar*. Si no, *rechazar*."

- En un DFA se marca desde el inicio **hacia adelante**.
- En una gramática, desde los terminales **hacia arriba**.
- $A \to \varepsilon$: $k = 0$, se marca enseguida.

- **Termina:** cada vuelta marca una variable o es la última. A lo sumo $|V|$ vueltas útiles.

E_{CFG}: marcar desde abajo

R = "Con la entrada $\langle G \rangle$, donde G es una gramática:

1. Marcar todos los terminales.
2. Repetir hasta no marcar nada nuevo: marcar toda variable A con una regla $A \to U_1 \cdots U_k$ cuyos símbolos estén todos marcados.
3. Si la variable inicial no quedó marcada, *aceptar*. Si no, *rechazar*."

- En un DFA se marca desde el inicio **hacia adelante**.
- En una gramática, desde los terminales **hacia arriba**.
- $A \to \varepsilon$: $k = 0$, se marca enseguida.

- **Termina:** cada vuelta marca una variable o es la última. A lo sumo $|V|$ vueltas útiles.
- **Correcta:** A queda marcada $\iff$ $A \Rightarrow^* x$ para alguna cadena x de terminales.
  - $\Rightarrow$: inducción sobre el orden de las marcas. $\Leftarrow$: inducción sobre los pasos de la derivación.

Una gramática que no genera nada

$$S \to ASB \mid CA$$
$$A \to \mathrm{a} \mid \mathrm{a}A$$
$$B \to \mathrm{b}$$
$$C \to C\mathrm{b} \mid S\mathrm{a}$$

Una recursión sin caso base: S necesita a S o a C, y C a C o a S.

terminal

```
$ ../build/e_cfg no_base_case.cfg
G = no_base_case.cfg, 4 variables, 7 reglas, inicial S
ronda 1:  se marca A por A → a;  se marca B por B → b;
ronda 2:  no se marca nada, termina
marcadas: 2 de 4
S no está marcada: L(G) = ∅, ⟨G⟩ ∈ E_CFG
```

- Desde S “se llega” a las cuatro variables. No dice nada.
- Importa si alguna derivación **termina**.

Por qué $\mathrm{EQ}_{\mathrm{CFG}}$ no sale igual

$$\mathrm{EQ}_{\mathrm{CFG}} = \{\, \langle G, H \rangle : G \text{ y } H \text{ son gramáticas y } L(G) = L(H) \,\}$$

- El plan de $\mathrm{EQ}_{\mathrm{DFA}}$: una gramática para $L(G) \triangle L(H)$, y después E_{CFG}.

Por qué $\mathrm{EQ}_{\mathrm{CFG}}$ no sale igual

$$\mathrm{EQ}_{\mathrm{CFG}} = \{\, \langle G, H \rangle : G \text{ y } H \text{ son gramáticas y } L(G) = L(H) \,\}$$

- El plan de $\mathrm{EQ}_{\mathrm{DFA}}$: una gramática para $L(G) \triangle L(H)$, y después E_{CFG}.
- Falla en el primer paso: los lenguajes libres de contexto **no** son cerrados bajo complemento ni bajo intersección (semana 6).
- $L(G) \triangle L(H)$ puede no tener ninguna gramática.

Por qué $\mathrm{EQ}_{\mathrm{CFG}}$ no sale igual

$$\mathrm{EQ}_{\mathrm{CFG}} = \{\, \langle G, H \rangle : G \text{ y } H \text{ son gramáticas y } L(G) = L(H) \,\}$$

- El plan de $\mathrm{EQ}_{\mathrm{DFA}}$: una gramática para $L(G) \triangle L(H)$, y después E_{CFG}.
- Falla en el primer paso: los lenguajes libres de contexto **no** son cerrados bajo complemento ni bajo intersección (semana 6).
- $L(G) \triangle L(H)$ puede no tener ninguna gramática.

Qué se concluye y qué no

Que una demostración no se pueda copiar **no** demuestra que el problema sea indecidible. En este caso sí lo es, y se demuestra en la semana 10.

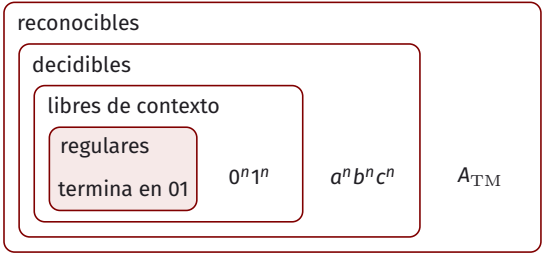
Todo lenguaje libre de contexto es decidible

M_G = "Con la entrada w:

1. Correr S con la entrada $\langle G, w \rangle$.
2. Si S acepta, *aceptar*. Si rechaza, *rechazar*."

G está fija, **dentro** de la máquina. S sirve para todas las gramáticas; hay una M_G por cada una.

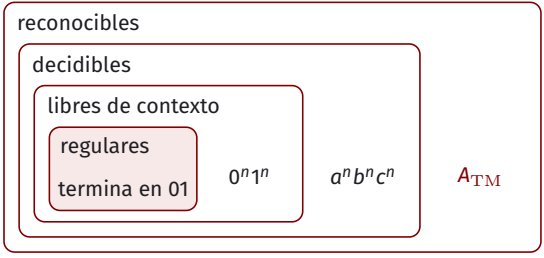
- Las tres inclusiones son estrictas.



Todo lenguaje libre de contexto es decidible

M_G = "Con la entrada w:

1. Correr S con la entrada $\langle G, w \rangle$.
2. Si S acepta, *aceptar*. Si rechaza, *rechazar*."



G está fija, **dentro** de la máquina. S sirve para todas las gramáticas; hay una M_G por cada una.

- Las tres inclusiones son estrictas. La última fila falta: es la sesión B.

Error común: “decidir” simulando algo que podría no detenerse

- Todas las máquinas de hoy tienen pasos como “simular B con w” o “correr T con $\langle C \rangle$”.

Error común: “decidir” simulando algo que podría no detenerse

- Todas las máquinas de hoy tienen pasos como “simular B con w” o “correr T con $\langle C \rangle$”.
- Son legítimos porque lo simulado **siempre termina**: un DFA da un paso por símbolo; T decide.

Error común: “decidir” simulando algo que podría no detenerse

- Todas las máquinas de hoy tienen pasos como “simular B con w” o “correr T con $\langle C \rangle$”.
- Son legítimos porque lo simulado **siempre termina**: un DFA da un paso por símbolo; T decide.
- “Simular la máquina de Turing M con w” no tiene esa garantía.

Antes de escribir “simular” o “correr”

¿Quién me asegura que esto se detiene?

Error común: “decidir” simulando algo que podría no detenerse

- Todas las máquinas de hoy tienen pasos como “simular B con w” o “correr T con $\langle C \rangle$”.
- Son legítimos porque lo simulado **siempre termina**: un DFA da un paso por símbolo; T decide.
- “Simular la máquina de Turing M con w” no tiene esa garantía.

Antes de escribir “simular” o “correr”

¿Quién me asegura que esto se detiene?

Para la sesión B: copien la máquina M de A_{DFA} cambiando “DFA” por “máquina de Turing”. ¿Qué parte de la demostración deja de valer?

SECCIÓN 05

La máquina universal

05



La misma pregunta, un modelo más arriba

$$A_{\mathrm{TM}} = \{\, \langle M, w \rangle : M \text{ es una máquina de Turing que acepta } w \,\}$$

U = "Con la entrada $\langle M, w \rangle$, donde M es una máquina de Turing y w una cadena:

1. Simular M con la entrada w.
2. Si M entra en su estado de aceptación, *aceptar*. Si entra en el de rechazo, *rechazar*."

La misma pregunta, un modelo más arriba

$$A_{\mathrm{TM}} = \{\, \langle M, w \rangle : M \text{ es una máquina de Turing que acepta } w \,\}$$

U = "Con la entrada $\langle M, w \rangle$, donde M es una máquina de Turing y w una cadena:

1. Simular M con la entrada w.
2. Si M entra en su estado de aceptación, *aceptar*. Si entra en el de rechazo, *rechazar*."

- ¿Existe U? Sí: simular es mantener una configuración y aplicar δ, que viene escrita en $\langle M \rangle$.

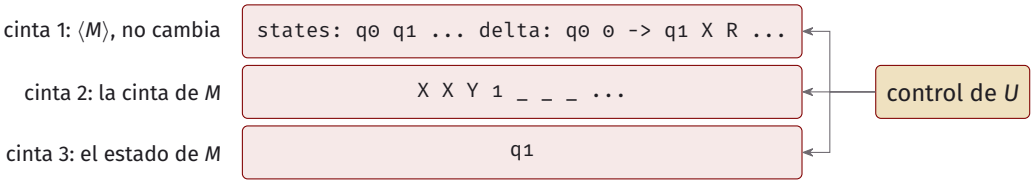
La misma pregunta, un modelo más arriba

$$A_{\mathrm{TM}} = \{\, \langle M, w \rangle : M \text{ es una máquina de Turing que acepta } w \,\}$$

> **U = "Con la entrada $\langle M, w \rangle$, donde M es una máquina de Turing y w una cadena:**
>
> 1. Simular M con la entrada w.
> 2. Si M entra en su estado de aceptación, *aceptar*. Si entra en el de rechazo, *rechazar*."

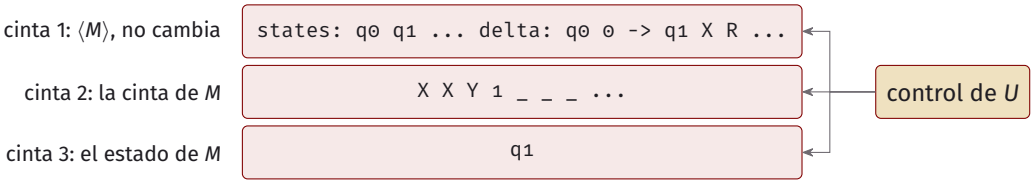
- ¿Existe U? Sí: simular es mantener una configuración y aplicar δ, que viene escrita en $\langle M \rangle$.
- Es **una sola** máquina que hace el trabajo de todas: la **máquina universal** *(universal Turing machine)*.
- Turing, 1936.

Un intérprete es una máquina universal



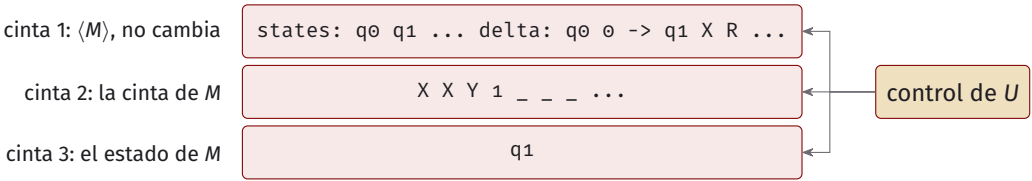
- Cada paso de M: buscar en la cinta 1 la transición para (estado, símbolo) y aplicarla. Tres cintas se simulan con una (semana 7).

Un intérprete es una máquina universal



- Cada paso de M: buscar en la cinta 1 la transición para (estado, símbolo) y aplicarla. Tres cintas se simulan con una (semana 7).
- El intérprete de Python, el que construyeron en Introducción a la Programación, un emulador, el procesador de su laptop: un programa fijo que recibe programas.

Un intérprete es una máquina universal



- Cada paso de M: buscar en la cinta 1 la transición para (estado, símbolo) y aplicarla. Tres cintas se simulan con una (semana 7).
- El intérprete de Python, el que construyeron en Introducción a la Programación, un emulador, el procesador de su laptop: un programa fijo que recibe programas.
- `universal.cpp` es U en C++: recibe un archivo `.tm` y una cadena.

La máquina es un argumento más

terminal

```
$ ../build/universal zeros_ones.tm 0011 --traza
⟨M⟩ = zeros_ones.tm: 6 estados, 11 transiciones, 296 caracteres
w = 0011
   q0 0011
  X q1 011
  X0 q1 11
  X q2 0Y1
   q2 X0Y1
  X q0 0Y1
  XX q1 Y1
  XXY q1 1
  XX q2 YY
  X q2 XYY
  XX q0 YY
  XXY q3 Y
  XXYY q3 _
  XXYY_ q_accept _
M acepta w después de 13 pasos
```

- $\langle M \rangle$ es una cadena de 296 caracteres.
- Cada línea es una configuración $u\ q\ v$.
- El programa no sabe nada de $\{0^n 1^n\}$: la máquina llegó como dato.
- Con `0010`, la misma máquina rechaza después de 9 pasos.

A_{TM} es reconocible

Teorema

U reconoce A_{TM}.

- Si *M* acepta *w*: lo hace en una cantidad finita *t* de pasos. *U* llega al paso *t* y **acepta**.

A_{TM} es reconocible

Teorema

U reconoce A_{TM}.

- Si M acepta w: lo hace en una cantidad finita t de pasos. U llega al paso t y **acepta**.
- Si M rechaza w: U lo ve y **rechaza**.

A_{TM} es reconocible

Teorema

U reconoce A_{TM}.

- Si *M* acepta *w*: lo hace en una cantidad finita *t* de pasos. *U* llega al paso *t* y **acepta**.
- Si *M* rechaza *w*: *U* lo ve y **rechaza**.
- Si *M* no se detiene con *w*: la simulación tampoco. *U* **no se detiene**; en particular, no acepta.

A_{TM} es reconocible

Teorema

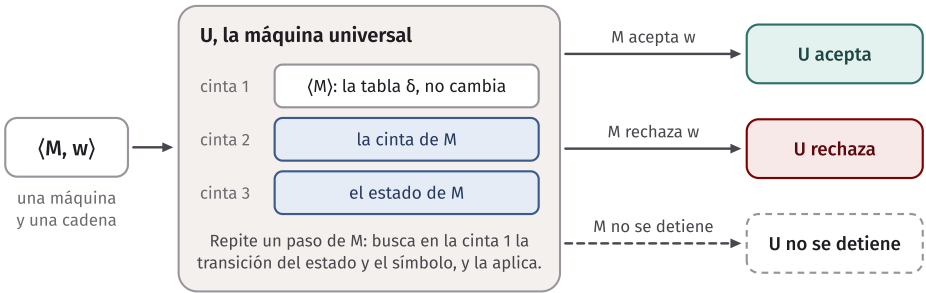
U reconoce A_{TM}.

- Si M acepta w: lo hace en una cantidad finita t de pasos. U llega al paso t y **acepta**.
- Si M rechaza w: U lo ve y **rechaza**.
- Si M no se detiene con w: la simulación tampoco. U **no se detiene**; en particular, no acepta.

Reconocible, no decidible

U acepta $\langle M, w \rangle$ si y solo si M acepta w. Pero hay entradas con las que U no se detiene.

U termina de tres maneras



Si M acepta o rechaza, lo hace después de una cantidad finita de pasos: U lo ve y responde lo mismo.

Si M no se detiene, la simulación tampoco. U acepta exactamente A_{TM}: lo reconoce, y no lo decide.

Por qué simular no decide

Dos máquinas para “contiene un 1”. Cada una tiene un solo estado de trabajo, *scan*:

	lee 0	lee 1	lee ␣
`has_one`	*scan*, R	acepta	**rechaza**
`search_one`	*scan*, R	acepta	*scan*, **R**

- Reconocen el **mismo** lenguaje.
- La primera lo decide. La segunda, sin un 1, sigue por los blancos para siempre.

terminal

```
$ ../build/universal search_one.tm 000 8 --traza
⟨M⟩ = search_one.tm: 3 estados, 3 transiciones, 171 caracteres
w = 000
   scan 000
  0 scan 00
  00 scan 0
  000 scan _
  000_ scan _
  000__ scan _
  000___ scan _
  000____ scan _
  000_____ scan _
M no se detuvo en 8 pasos
```

El 8 es un límite de pasos.

Dos arreglos que no funcionan

1. Subir el límite

terminal

```
$ ../build/universal search_one.tm 000 1000000
⟨M⟩ = search_one.tm: 3 estados, 3 transiciones, 171 caracteres
w = 000
M no se detuvo en 1000000 pasos
$ ../build/universal zeros_ones.tm 0011 5
⟨M⟩ = zeros_ones.tm: 6 estados, 11 transiciones, 296 caracteres
w = 0011
M no se detuvo en 5 pasos
```

La misma respuesta para una que no para y para una que acepta en 13 pasos. “No se detuvo en *N* pasos” habla de *M*, de *w* **y de** *N*.

Dos arreglos que no funcionan

1. Subir el límite

terminal

```
$ ../build/universal search_one.tm 000 1000000
⟨M⟩ = search_one.tm: 3 estados, 3 transiciones, 171 caracteres
w = 000
M no se detuvo en 1000000 pasos
$ ../build/universal zeros_ones.tm 0011 5
⟨M⟩ = zeros_ones.tm: 6 estados, 11 transiciones, 296 caracteres
w = 0011
M no se detuvo en 5 pasos
```

La misma respuesta para una que no para y para una que acepta en 13 pasos. “No se detuvo en *N* pasos” habla de *M*, de *w* **y de** *N*.

2. Detectar que se repite

- Si una configuración se repite, no se detiene. Cierto.
- Pero puede no detenerse **sin repetir ninguna**: el cabezal de `search_one.tm` está cada vez en otra celda.

Dos arreglos que no funcionan

1. Subir el límite

terminal

```
$ ../build/universal search_one.tm 000 1000000
⟨M⟩ = search_one.tm: 3 estados, 3 transiciones, 171 caracteres
w = 000
M no se detuvo en 1000000 pasos
$ ../build/universal zeros_ones.tm 0011 5
⟨M⟩ = zeros_ones.tm: 6 estados, 11 transiciones, 296 caracteres
w = 0011
M no se detuvo en 5 pasos
```

La misma respuesta para una que no para y para una que acepta en 13 pasos. “No se detuvo en *N* pasos” habla de *M*, de *w* **y de** *N*.

2. Detectar que se repite

- Si una configuración se repite, no se detiene. Cierto.
- Pero puede no detenerse **sin repetir ninguna**: el cabezal de `search_one.tm` está cada vez en otra celda.

Probar ideas no descarta **todas** las ideas. Hace falta demostrar una imposibilidad.

SECCIÓN 06

La diagonal de Cantor

06

Conjuntos numerables

Numerable *(countable)*

Finito, o en correspondencia uno a uno con $1, 2, 3, \ldots$
Es decir: sus elementos se pueden poner en una **lista** en la que cada uno aparece en alguna posición finita.

- Los pares: $2, 4, 6, \ldots$ Los enteros: $0, 1, -1, 2, -2, \ldots$

Conjuntos numerables

Numerable *(countable)*

Finito, o en correspondencia uno a uno con $1, 2, 3, \ldots$
Es decir: sus elementos se pueden poner en una **lista** en la que cada uno aparece en alguna posición finita.

- Los pares: $2, 4, 6, \ldots$ Los enteros: $0, 1, -1, 2, -2, \ldots$
- Los pares (i, j) de naturales: por diagonales, primero los que suman 2, después los que suman 3…

Conjuntos numerables

Numerable *(countable)*

Finito, o en correspondencia uno a uno con $1, 2, 3, \ldots$
Es decir: sus elementos se pueden poner en una **lista** en la que cada uno aparece en alguna posición finita.

- Los pares: $2, 4, 6, \ldots$ Los enteros: $0, 1, -1, 2, -2, \ldots$
- Los pares (i, j) de naturales: por diagonales, primero los que suman 2, después los que suman 3…
- Σ^*: primero las cadenas más cortas. Es la lista de la semana 1.

Conjuntos numerables

Numerable *(countable)*

Finito, o en correspondencia uno a uno con $1, 2, 3, \ldots$
Es decir: sus elementos se pueden poner en una **lista** en la que cada uno aparece en alguna posición finita.

- Los pares: $2, 4, 6, \ldots$ Los enteros: $0, 1, -1, 2, -2, \ldots$
- Los pares (i, j) de naturales: por diagonales, primero los que suman 2, después los que suman 3…
- Σ^*: primero las cadenas más cortas. Es la lista de la semana 1.
- Un subconjunto de un numerable es numerable: se tachan los que no están.

Repaso de Matemática Discreta, en una página de los apuntes.

La diagonal de Cantor: los reales no son numerables

$r_1 = 0,$	**1**	4	1	5	9	$\cdots$
$r_2 = 0,$	5	**0**	0	0	0	$\cdots$
$r_3 = 0,$	3	3	**3**	3	3	$\cdots$
$r_4 = 0,$	7	1	8	**2**	8	$\cdots$
$r_5 = 0,$	0	4	4	4	**4**	$\cdots$

Una lista de ejemplo.

Supongamos una lista con **todos** los reales entre 0 y 1.

La diagonal de Cantor: los reales no son numerables

$r_1 = 0,$	1	4	1	5	9	$\cdots$
$r_2 = 0,$	5	0	0	0	0	$\cdots$
$r_3 = 0,$	3	3	3	3	3	$\cdots$
$r_4 = 0,$	7	1	8	2	8	$\cdots$
$r_5 = 0,$	0	4	4	4	4	$\cdots$
$x = 0,$	**4**	**4**	**4**	**4**	**5**	$\cdots$

Una lista de ejemplo.

Supongamos una lista con **todos** los reales entre 0 y 1.

- El decimal n de x es **distinto** del decimal n de r_n: un 5 si aquel es 4, y un 4 si no.

La diagonal de Cantor: los reales no son numerables

$r_1 = 0,$	1	4	1	5	9	$\cdots$
$r_2 = 0,$	5	0	0	0	0	$\cdots$
$r_3 = 0,$	3	3	3	3	3	$\cdots$
$r_4 = 0,$	7	1	8	2	8	$\cdots$
$r_5 = 0,$	0	4	4	4	4	$\cdots$
$x = 0,$	**4**	**4**	**4**	**4**	**5**	$\cdots$

Una lista de ejemplo.

Supongamos una lista con **todos** los reales entre 0 y 1.

- El decimal n de x es **distinto** del decimal n de r_n: un 5 si aquel es 4, y un 4 si no.
- $x \neq r_n$ para todo n: difieren en el decimal n.
- x no está en la lista. Contradicción.

La diagonal de Cantor: los reales no son numerables

$r_1 = 0,$	1	4	1	5	9	$\cdots$
$r_2 = 0,$	5	0	0	0	0	$\cdots$
$r_3 = 0,$	3	3	3	3	3	$\cdots$
$r_4 = 0,$	7	1	8	2	8	$\cdots$
$r_5 = 0,$	0	4	4	4	4	$\cdots$
$x = 0,$	**4**	**4**	**4**	**4**	**5**	$\cdots$

Una lista de ejemplo.

Supongamos una lista con **todos** los reales entre 0 y 1.

- El decimal n de x es **distinto** del decimal n de r_n: un 5 si aquel es 4, y un 4 si no.
- $x \neq r_n$ para todo n: difieren en el decimal n.
- x no está en la lista. Contradicción.

El método

Una lista, una tabla, y un objeto nuevo que en la posición n hace lo contrario de la casilla (n, n).

Hay más lenguajes que máquinas

- **Las máquinas de Turing son numerables.** Cada una es una cadena $\langle M \rangle$. De la lista de todas las cadenas se tachan las que no describen una máquina: queda $M_1, M_2, M_3, \ldots$

Hay más lenguajes que máquinas

- **Las máquinas de Turing son numerables.** Cada una es una cadena $\langle M \rangle$. De la lista de todas las cadenas se tachan las que no describen una máquina: queda $M_1, M_2, M_3, \ldots$
- **Los lenguajes no.** Dada una lista $L_1, L_2, \ldots$ y las cadenas $s_1, s_2, \ldots$:

$$D = \{ s_i : s_i \notin L_i \} \quad \text{difiere de } L_i \text{ en } s_i.$$

Hay más lenguajes que máquinas

- **Las máquinas de Turing son numerables.** Cada una es una cadena $\langle M \rangle$. De la lista de todas las cadenas se tachan las que no describen una máquina: queda $M_1, M_2, M_3, \ldots$
- **Los lenguajes no.** Dada una lista $L_1, L_2, \ldots$ y las cadenas $s_1, s_2, \ldots$:

$$D = \{ s_i : s_i \notin L_i \} \quad \text{difiere de } L_i \text{ en } s_i.$$

- Cada máquina reconoce **un** lenguaje. Los reconocibles son $L(M_1), L(M_2), \ldots$: numerables.

Hay más lenguajes que máquinas

- **Las máquinas de Turing son numerables.** Cada una es una cadena $\langle M \rangle$. De la lista de todas las cadenas se tachan las que no describen una máquina: queda $M_1, M_2, M_3, \ldots$
- **Los lenguajes no.** Dada una lista $L_1, L_2, \ldots$ y las cadenas $s_1, s_2, \ldots$:

$$D = \{ s_i : s_i \notin L_i \} \quad \text{difiere de } L_i \text{ en } s_i.$$

- Cada máquina reconoce **un** lenguaje. Los reconocibles son $L(M_1), L(M_2), \ldots$: numerables.

Corolario

Existen lenguajes que ninguna máquina de Turing reconoce. De hecho, casi todos.

Hay más lenguajes que máquinas

- **Las máquinas de Turing son numerables.** Cada una es una cadena $\langle M \rangle$. De la lista de todas las cadenas se tachan las que no describen una máquina: queda $M_1, M_2, M_3, \ldots$
- **Los lenguajes no.** Dada una lista $L_1, L_2, \ldots$ y las cadenas $s_1, s_2, \ldots$:

$$D = \{ s_i : s_i \notin L_i \} \quad \text{difiere de } L_i \text{ en } s_i.$$

- Cada máquina reconoce **un** lenguaje. Los reconocibles son $L(M_1), L(M_2), \ldots$: numerables.

Corolario

Existen lenguajes que ninguna máquina de Turing reconoce. De hecho, casi todos.

El mismo defecto que en la semana 1: no dice **cuál**. D no se parece a nada útil.

SECCIÓN 07

A_{TM} es indecidible

07

El teorema

Teorema

A_{TM} es indecidible: ninguna máquina de Turing, con toda entrada $\langle M, w \rangle$, se detiene y responde correctamente si M acepta w.

Por contradicción. Supongamos que una máquina H decide A_{TM}:

$$H(\langle M, w \rangle) = \begin{cases} \text{acepta} & \text{si } M \text{ acepta } w \\ \text{rechaza} & \text{si } M \text{ no acepta } w \end{cases}$$

El teorema

Teorema

A_{TM} es indecidible: ninguna máquina de Turing, con toda entrada $\langle M, w \rangle$, se detiene y responde correctamente si M acepta w.

Por contradicción. Supongamos que una máquina H decide A_{TM}:

$$H(\langle M, w \rangle) = \begin{cases} \text{acepta} & \text{si } M \text{ acepta } w \\ \text{rechaza} & \text{si } M \text{ no acepta } w \end{cases}$$

- "No acepta" incluye dos casos: M rechaza w, o M no se detiene con w.

El teorema

Teorema

A_{TM} es indecidible: ninguna máquina de Turing, con toda entrada $\langle M, w \rangle$, se detiene y responde correctamente si M acepta w.

Por contradicción. Supongamos que una máquina H decide A_{TM}:

$$H(\langle M, w \rangle) = \begin{cases} \text{acepta} & \text{si } M \text{ acepta } w \\ \text{rechaza} & \text{si } M \text{ no acepta } w \end{cases}$$

- "No acepta" incluye dos casos: M rechaza w, o M no se detiene con w.
- La diferencia con U: H **siempre se detiene**, también cuando M no.

La máquina D

D = "Con la entrada $\langle M \rangle$, donde M es una máquina de Turing:

1. Correr H con la entrada $\langle M, \langle M \rangle \rangle$.
2. Si H acepta, *rechazar*. Si H rechaza, *aceptar*."

- D le pregunta a H qué hace M **con su propia descripción**, y hace lo contrario.

La máquina D

D = "Con la entrada $\langle M \rangle$, donde M es una máquina de Turing:

1. Correr H con la entrada $\langle M, \langle M \rangle \rangle$.
2. Si H acepta, *rechazar*. Si H rechaza, *aceptar*."

- D le pregunta a H qué hace M **con su propia descripción**, y hace lo contrario.
- Aquí se usa que H siempre se detiene: el paso 1 termina, así que D siempre llega al paso 2 y responde.

La máquina D

D = "Con la entrada $\langle M \rangle$, donde M es una máquina de Turing:

1. Correr H con la entrada $\langle M, \langle M \rangle \rangle$.
2. Si H acepta, *rechazar*. Si H rechaza, *aceptar*."

- D le pregunta a H qué hace M **con su propia descripción**, y hace lo contrario.
- Aquí se usa que H siempre se detiene: el paso 1 termina, así que D siempre llega al paso 2 y responde.

$$D(\langle M \rangle) = \begin{cases} \text{acepta} & \text{si } M \text{ no acepta } \langle M \rangle \\ \text{rechaza} & \text{si } M \text{ acepta } \langle M \rangle \end{cases}$$

D con la entrada $\langle D \rangle$

D es una máquina de Turing: tiene una descripción $\langle D \rangle$, y es una entrada válida para D.

$$D(\langle D \rangle) = \begin{cases} \text{acepta} & \text{si } D \text{ no acepta } \langle D \rangle \\ \text{rechaza} & \text{si } D \text{ acepta } \langle D \rangle \end{cases}$$

D con la entrada $\langle D \rangle$

D es una máquina de Turing: tiene una descripción $\langle D \rangle$, y es una entrada válida para D.

$$D(\langle D \rangle) = \begin{cases} \text{acepta} & \text{si } D \text{ no acepta } \langle D \rangle \\ \text{rechaza} & \text{si } D \text{ acepta } \langle D \rangle \end{cases}$$

- **Si D acepta $\langle D \rangle$:** H acepta $\langle D, \langle D \rangle \rangle$, y por el paso 2, D rechaza $\langle D \rangle$. Imposible.

D con la entrada $\langle D \rangle$

D es una máquina de Turing: tiene una descripción $\langle D \rangle$, y es una entrada válida para D.

$$D(\langle D \rangle) = \begin{cases} \text{acepta} & \text{si } D \text{ no acepta } \langle D \rangle \\ \text{rechaza} & \text{si } D \text{ acepta } \langle D \rangle \end{cases}$$

- **Si D acepta $\langle D \rangle$:** H acepta $\langle D, \langle D \rangle \rangle$, y por el paso 2, D rechaza $\langle D \rangle$. Imposible.
- **Si D no acepta $\langle D \rangle$:** H rechaza $\langle D, \langle D \rangle \rangle$, y por el paso 2, D acepta $\langle D \rangle$. Imposible.

D con la entrada $\langle D \rangle$

D es una máquina de Turing: tiene una descripción $\langle D \rangle$, y es una entrada válida para D.

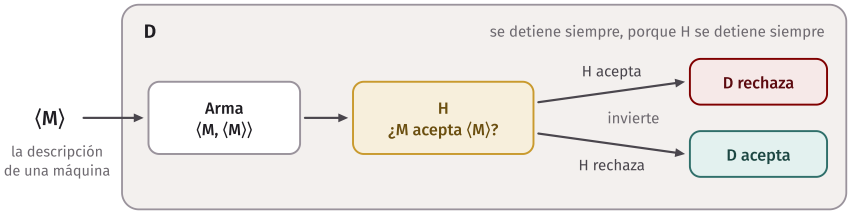
$$D(\langle D \rangle) = \begin{cases} \text{acepta} & \text{si } D \text{ no acepta } \langle D \rangle \\ \text{rechaza} & \text{si } D \text{ acepta } \langle D \rangle \end{cases}$$

- **Si D acepta $\langle D \rangle$:** H acepta $\langle D, \langle D \rangle \rangle$, y por el paso 2, D rechaza $\langle D \rangle$. Imposible.
- **Si D no acepta $\langle D \rangle$:** H rechaza $\langle D, \langle D \rangle \rangle$, y por el paso 2, D acepta $\langle D \rangle$. Imposible.

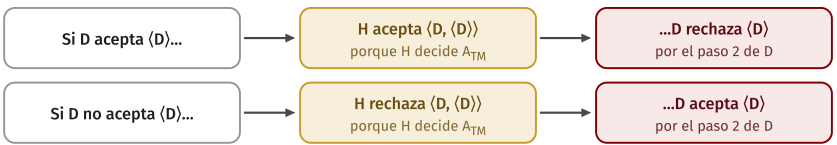
Contradicción

D acepta $\langle D \rangle$ si y solo si D no acepta $\langle D \rangle$. El único supuesto fue que H existe. H **no existe.**

La demostración en un dibujo

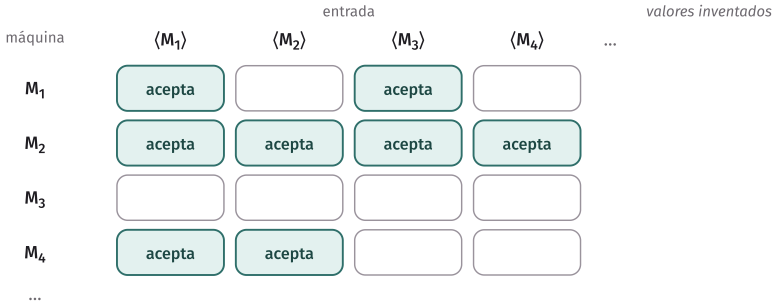


¿Qué hace D con la entrada ⟨D⟩? Es el mismo diagrama con M = D.



En los dos casos se llega a lo contrario de lo supuesto. Lo único supuesto fue que H existe: H no existe.

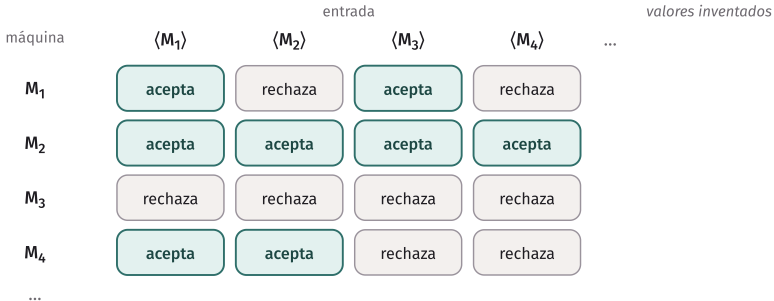
Dónde está la diagonal



La casilla (i, j) dice si M_i acepta $\langle M_j \rangle$. Una casilla vacía: no acepta (rechaza o no se detiene).

Los valores son inventados, para ver la forma. La tabla existe aunque nadie pueda calcularla.

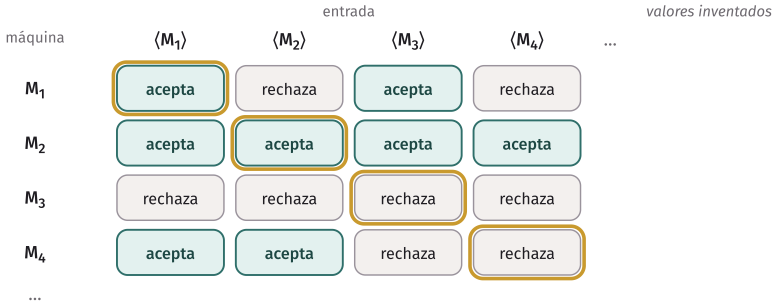
Dónde está la diagonal



Si H existiera, calcularía cualquier casilla: es su respuesta con ⟨M_i, ⟨M_j⟩⟩.

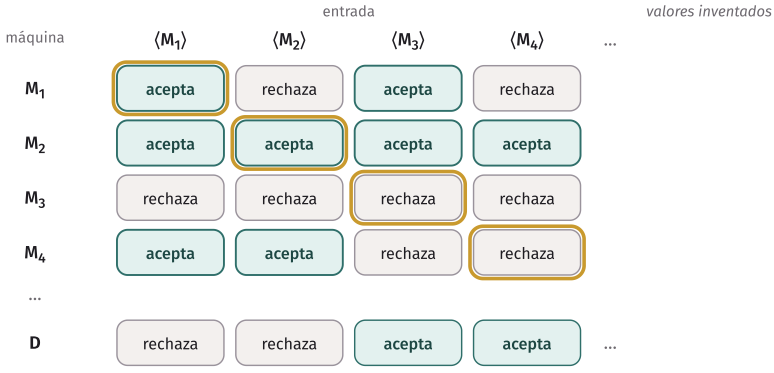
En las casillas vacías H respondería «rechaza», también cuando M_i no se detiene.

Dónde está la diagonal



La diagonal: la casilla (i, i) dice qué hace M_i con su propia descripción.

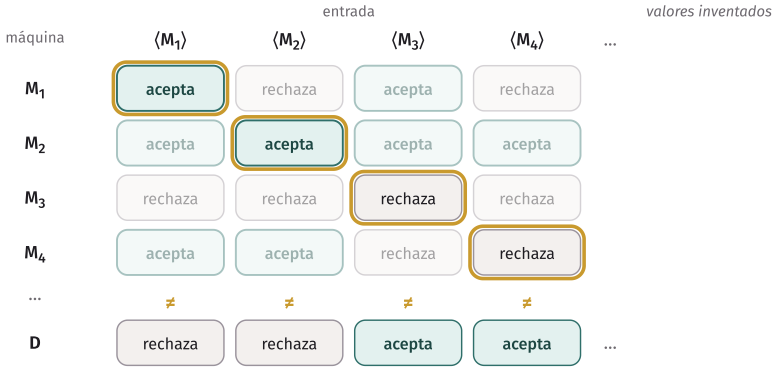
Dónde está la diagonal



D, con la entrada $\langle M_i \rangle$, mira la casilla (i, i) y hace lo contrario.

Su fila sale de invertir la diagonal: rechaza, rechaza, acepta, acepta.

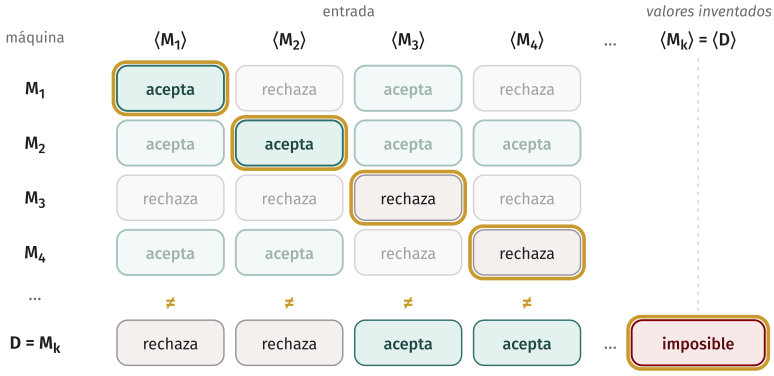
Dónde está la diagonal



La fila de D difiere de la de M_1 en la columna 1, de la de M_2 en la 2, y de la de M_i en la columna i.

Entonces D no es ninguna máquina de la lista.

Dónde está la diagonal



Pero D es una máquina de Turing, y la lista las tiene a todas: D es M_k para algún k.

En la casilla (k, k), D con la entrada ⟨D⟩, la fila tendría que ser distinta de sí misma.

El paso que usa "H siempre se detiene"

Pongamos en lugar de H la máquina universal U, que sí existe:

> D' = **"Con la entrada** $\langle M \rangle$**:**
>
> 1. Correr U con la entrada $\langle M, \langle M \rangle \rangle$.
> 2. Si U acepta, *rechazar*. Si U rechaza, *aceptar*."

El paso que usa "H siempre se detiene"

Pongamos en lugar de H la máquina universal U, que sí existe:

> D' = **"Con la entrada $\langle M \rangle$:**
>
> 1. Correr U con la entrada $\langle M, \langle M \rangle \rangle$.
> 2. Si U acepta, *rechazar*. Si U rechaza, *aceptar*."

- D' con $\langle D' \rangle$: U simula a D' con $\langle D' \rangle$, que empieza simulando a D' con $\langle D' \rangle$… **No se detiene.**

El paso que usa "H siempre se detiene"

Pongamos en lugar de H la máquina universal U, que sí existe:

> D' = **"Con la entrada $\langle M \rangle$:**
>
> 1. Correr U con la entrada $\langle M, \langle M \rangle \rangle$.
> 2. Si U acepta, *rechazar*. Si U rechaza, *aceptar*."

- D' con $\langle D' \rangle$: U simula a D' con $\langle D' \rangle$, que empieza simulando a D' con $\langle D' \rangle$… **No se detiene.**
- Y no hay contradicción: D' no acepta $\langle D' \rangle$, y U efectivamente no acepta $\langle D', \langle D' \rangle \rangle$.

El paso que usa "H siempre se detiene"

Pongamos en lugar de H la máquina universal U, que sí existe:

> D' = **"Con la entrada $\langle M \rangle$:**
>
> 1. Correr U con la entrada $\langle M, \langle M \rangle \rangle$.
> 2. Si U acepta, *rechazar*. Si U rechaza, *aceptar*."

- D' con $\langle D' \rangle$: U simula a D' con $\langle D' \rangle$, que empieza simulando a D' con $\langle D' \rangle$… **No se detiene.**
- Y no hay contradicción: D' no acepta $\langle D' \rangle$, y U efectivamente no acepta $\langle D', \langle D' \rangle \rangle$.
- La contradicción nace solo cuando el paso 1 está **obligado a terminar con una respuesta**. Eso prometía H, y U no.

Lo que la demostración es, y lo que no

- **Darle a una máquina su propia descripción es legal.** $\langle M \rangle$ es una cadena como cualquier otra. Un compilador de C++ escrito en C++ se compila a sí mismo.

Lo que la demostración es, y lo que no

- **Darle a una máquina su propia descripción es legal.** $\langle M \rangle$ es una cadena como cualquier otra. Un compilador de C++ escrito en C++ se compila a sí mismo.
- **No depende de detalles.** Vale para cualquier H, y por Church-Turing para cualquier lenguaje de programación.

Lo que la demostración es, y lo que no

- **Darle a una máquina su propia descripción es legal.** $\langle M \rangle$ es una cadena como cualquier otra. Un compilador de C++ escrito en C++ se compila a sí mismo.
- **No depende de detalles.** Vale para cualquier H, y por Church-Turing para cualquier lenguaje de programación.
- **No es una paradoja.** La contradicción depende de un supuesto, y se concluye un teorema: H no existe.

Lo que la demostración es, y lo que no

- **Darle a una máquina su propia descripción es legal.** $\langle M \rangle$ es una cadena como cualquier otra. Un compilador de C++ escrito en C++ se compila a sí mismo.
- **No depende de detalles.** Vale para cualquier H, y por Church-Turing para cualquier lenguaje de programación.
- **No es una paradoja.** La contradicción depende de un supuesto, y se concluye un teorema: H no existe.

Error común

Leerlo como "todavía nadie encontró el algoritmo". No es una pregunta abierta: está **demostrado** que no existe, como está demostrado que $\sqrt{2}$ no es una fracción.

SECCIÓN 08

La parada y lo no reconocible

08

El problema de la parada

$$\mathrm{HALT}_{\mathrm{TM}} = \{\, \langle M, w \rangle : M \text{ se detiene con la entrada } w \,\}$$

Indecidible. Reducción desde A_{TM} hacia $\mathrm{HALT}_{\mathrm{TM}}$: si R decidiera $\mathrm{HALT}_{\mathrm{TM}}$, esta máquina decidiría A_{TM}.

> S = **"Con la entrada** $\langle M, w \rangle$**:**
>
> 1. Correr R con la entrada $\langle M, w \rangle$. Si R rechaza, *rechazar*.
> 2. Si R acepta, simular M con w hasta que se detenga.
> 3. Si M aceptó, *aceptar*. Si M rechazó, *rechazar*."

El problema de la parada

$$\mathrm{HALT}_{\mathrm{TM}} = \{\, \langle M, w \rangle : M \text{ se detiene con la entrada } w \,\}$$

Indecidible. Reducción desde A_{TM} hacia $\mathrm{HALT}_{\mathrm{TM}}$: si R decidiera $\mathrm{HALT}_{\mathrm{TM}}$, esta máquina decidiría A_{TM}.

> S = **"Con la entrada** $\langle M, w \rangle$**:**
>
> 1. Correr R con la entrada $\langle M, w \rangle$. Si R rechaza, *rechazar*.
> 2. Si R acepta, simular M con w hasta que se detenga.
> 3. Si M aceptó, *aceptar*. Si M rechazó, *rechazar*."

- El paso 2 es una simulación **con garantía**: solo se llega si M se detiene.

El problema de la parada

$$\mathrm{HALT}_{\mathrm{TM}} = \{\, \langle M, w \rangle : M \text{ se detiene con la entrada } w \,\}$$

Indecidible. Reducción desde A_{TM} hacia $\mathrm{HALT}_{\mathrm{TM}}$: si R decidiera $\mathrm{HALT}_{\mathrm{TM}}$, esta máquina decidiría A_{TM}.

> S = **"Con la entrada $\langle M, w \rangle$:**
>
> 1. Correr R con la entrada $\langle M, w \rangle$. Si R rechaza, *rechazar*.
> 2. Si R acepta, simular M con w hasta que se detenga.
> 3. Si M aceptó, *aceptar*. Si M rechazó, *rechazar*."

- El paso 2 es una simulación **con garantía**: solo se llega si M se detiene.
- A U solo le faltaba saber si la simulación termina: es el mismo fenómeno. Reducciones: semana 10.

Decidible = reconocible + correconocible

Teorema

L es decidible si y solo si L y su complemento $\overline{L}$ son reconocibles.

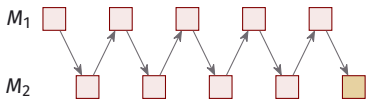
- $\Rightarrow$: un decididor reconoce;
  intercambiando aceptar y rechazar,
  decide $\overline{L}$.

Decidible = reconocible + correconocible

Teorema

L es decidible si y solo si L y su complemento $\overline{L}$ son reconocibles.

- $\Rightarrow$: un decididor reconoce; intercambiando aceptar y rechazar, decide $\overline{L}$.
- $\Leftarrow$: M_1 reconoce L y M_2 reconoce $\overline{L}$. Se corren **alternando** un paso de cada una.



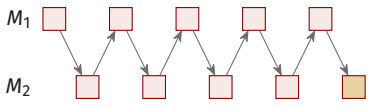
Si M_1 acepta, *aceptar*.
Si M_2 acepta, *rechazar*.

Decidible = reconocible + correconocible

Teorema

L es decidible si y solo si L y su complemento $\overline{L}$ son reconocibles.

- $\Rightarrow$: un decididor reconoce; intercambiando aceptar y rechazar, decide $\overline{L}$.
- $\Leftarrow$: M_1 reconoce L y M_2 reconoce $\overline{L}$. Se corren **alternando** un paso de cada una.
- Toda w está en L o en $\overline{L}$: alguna de las dos acepta, en finitos pasos.



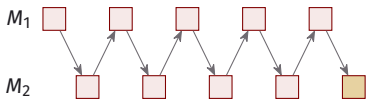
Si M_1 acepta, *aceptar*.
Si M_2 acepta, *rechazar*.

Decidible = reconocible + correconocible

Teorema

L es decidible si y solo si L y su complemento $\overline{L}$ son reconocibles.

- $\Rightarrow$: un decididor reconoce; intercambiando aceptar y rechazar, decide $\overline{L}$.
- $\Leftarrow$: M_1 reconoce L y M_2 reconoce $\overline{L}$. Se corren **alternando** un paso de cada una.
- Toda w está en L o en $\overline{L}$: alguna de las dos acepta, en finitos pasos.
- Correr M_1 hasta que termine y después M_2 **no sirve**: M_1 puede no detenerse.



Si M_1 acepta, *aceptar*.
Si M_2 acepta, *rechazar*.

Un lenguaje que no es reconocible

terminal

```
$ ../build/alternate search_one.tm search_end.tm 0001 000 '' 1
w = 0001: M1 acepta después de 4 rondas; se acepta
w = 000: M2 acepta después de 4 rondas; se rechaza
w = ε: M2 acepta después de 1 ronda; se rechaza
w = 1: M1 acepta después de 1 ronda; se acepta
```

Dos máquinas que reconocen sin decidir: “contiene un 1” y su complemento. Alternadas, deciden.

Un lenguaje que no es reconocible

terminal

```
$ ../build/alternate search_one.tm search_end.tm 0001 000 '' 1
w = 0001: M1 acepta después de 4 rondas; se acepta
w = 000: M2 acepta después de 4 rondas; se rechaza
w = ε: M2 acepta después de 1 ronda; se rechaza
w = 1: M1 acepta después de 1 ronda; se acepta
```

Dos máquinas que reconocen sin decidir: “contiene un 1” y su complemento. Alternadas, deciden.

Corolario

$\overline{A_{\mathrm{TM}}}$ **no es reconocible**: A_{TM} lo es, y si $\overline{A_{\mathrm{TM}}}$ también lo fuera, A_{TM} sería decidible.

Un lenguaje que no es reconocible

terminal

```
$ ../build/alternate search_one.tm search_end.tm 0001 000 '' 1
w = 0001: M1 acepta después de 4 rondas; se acepta
w = 000: M2 acepta después de 4 rondas; se rechaza
w = ε: M2 acepta después de 1 ronda; se rechaza
w = 1: M1 acepta después de 1 ronda; se acepta
```

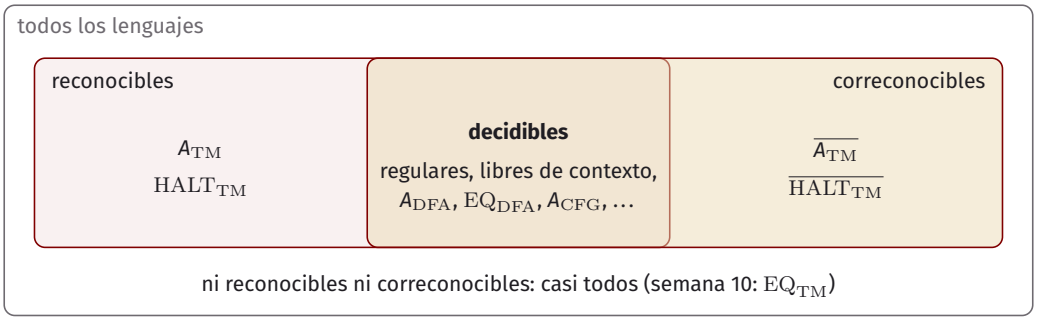
Dos máquinas que reconocen sin decidir: “contiene un 1” y su complemento. Alternadas, deciden.

Corolario

$\overline{A_{\mathrm{TM}}}$ **no es reconocible**: A_{TM} lo es, y si $\overline{A_{\mathrm{TM}}}$ también lo fuera, A_{TM} sería decidible.

Que M acepta w se puede **confirmar** mostrando la ejecución, que es finita. Que no la acepta porque no se detiene no tiene ninguna confirmación finita.

El mapa, completo



- Los decidibles son exactamente la zona donde los dos conjuntos se cruzan: es lo que dice el teorema.

Qué significa en la práctica

No existe un programa que reciba cualquier programa, con su entrada, y diga siempre correctamente si termina.

Qué significa en la práctica

No existe un programa que reciba cualquier programa, con su entrada, y diga siempre correctamente si termina.

- Sí se puede: analizar **un** programa concreto; restringir la clase (para los DFA todo es decidible).

Qué significa en la práctica

No existe un programa que reciba cualquier programa, con su entrada, y diga siempre correctamente si termina.

- Sí se puede: analizar **un** programa concreto; restringir la clase (para los DFA todo es decidible).
- Sí se puede: responder “termina”, “no termina” o **“no sé”**; avisar de más; poner un límite.
- El juez no decide si su programa termina: decide si termina **en 2 segundos**.

Qué significa en la práctica

No existe un programa que reciba cualquier programa, con su entrada, y diga siempre correctamente si termina.

- Sí se puede: analizar **un** programa concreto; restringir la clase (para los DFA todo es decidible).
- Sí se puede: responder "termina", "no termina" o **"no sé"**; avisar de más; poner un límite.
- El juez no decide si su programa termina: decide si termina **en 2 segundos**.

Error común

"No se puede saber nada de ningún programa." Indecidible quiere decir que ningún algoritmo acierta en **todos** los casos.

Qué significa en la práctica

No existe un programa que reciba cualquier programa, con su entrada, y diga siempre correctamente si termina.

- Sí se puede: analizar **un** programa concreto; restringir la clase (para los DFA todo es decidible).
- Sí se puede: responder "termina", "no termina" o **"no sé"**; avisar de más; poner un límite.
- El juez no decide si su programa termina: decide si termina **en 2 segundos**.

Error común

"No se puede saber nada de ningún programa." Indecidible quiere decir que ningún algoritmo acierta en **todos** los casos.

"Mi detector de ciclos acertó con mil programas": puede ser. El teorema garantiza que hay uno con el que falla, y *D* dice cómo construirlo.

Laboratorio: ¿son equivalentes dos DFA?

1. Leer dos DFA, separados por una línea `---`, y responder `equivalentes` o `distintos`.
2. Si son distintos, imprimir una cadena de longitud mínima que uno acepta y el otro no (`eps` si es la vacía).
3. Recorrer solo los pares alcanzables, en anchura.
4. Pasar los casos grandes: 50 000 estados cada uno, 2 500 000 000 de pares, 50 000 alcanzables.
5. Comparar el DFA de la semana 2 para “el sexto desde el final es 1” con su versión mínima de la semana 4.

Dónde: `code/week-09/lab/dfa_equiv`. Hay varias cadenas correctas: el juez usa un verificador.

`casos/01.out`

```
distintos
0101
```

`casos/03.out`

```
distintos
eps
```

`casos/02.out`

```
equivalentes
```

Para la próxima clase

- Leer los apuntes de la semana 9 y terminar el laboratorio.
- Escribir, sin mirar, la demostración de que A_{TM} es indecidible. ¿En qué paso se usa que *H* siempre se detiene?
- Preguntas de repaso de la guía: un algoritmo para “el DFA acepta alguna cadena de longitud par”; por qué el argumento de $\mathrm{EQ}_{\mathrm{DFA}}$ no sirve para $\mathrm{EQ}_{\mathrm{CFG}}$; qué responderle al compañero del detector de ciclos.
- Próxima semana: **reducciones**. Con A_{TM} como punto de partida, una familia entera de problemas indecidibles, el teorema de Rice y la deuda de $\mathrm{EQ}_{\mathrm{CFG}}$.

Pregunta

¿Es decidible si el lenguaje de una máquina de Turing es vacío? Intenten adaptar la máquina *T* de E_{DFA} y vean dónde se rompe.