

UNIVERSIDAD NACIONAL DE INGENIERÍA

Teoría de la Computación

Lo que un autómata finito no puede hacer

Lema de bombeo, cadenas distinguibles, Myhill-Nerode y minimización

Gerson Garrido

Semana 4

Hoy

Sesión A

1. Contar sin tope: $\{0^n 1^n\}$
2. Un estado que se repite
3. El lema de bombeo
4. El lema como un juego
5. Cinco demostraciones
6. Cierre, y lo que el lema no dice

Sesión B

7. Cadenas distinguibles
8. La deuda de la semana 2: 2^k estados
9. El teorema de Myhill-Nerode
10. Minimizar un DFA
11. El DFA mínimo es único
12. Opcional: memoria en *streaming*

Lectura

Apuntes de la semana 4. Apoyo: Sipser, sec. 1.4; Erickson, *Models of Computation*, nota 3, sección sobre *fooling sets*. Código en `code/week-04`.

SECCIÓN 01

Contar sin tope

01

Tres semanas, una sola clase

- Semana 1: los lenguajes que reconoce algún **DFA**.
- Semana 2: un **NFA** no reconoce nada más.
- Semana 3: las **expresiones regulares** describen lo mismo.
- Unión, intersección, complemento, concatenación, estrella: no sacan de la clase.

Tres semanas, una sola clase

- Semana 1: los lenguajes que reconoce algún **DFA**.
- Semana 2: un **NFA** no reconoce nada más.
- Semana 3: las **expresiones regulares** describen lo mismo.
- Unión, intersección, complemento, concatenación, estrella: no sacan de la clase.

¿Entonces todo lenguaje razonable es regular? No.

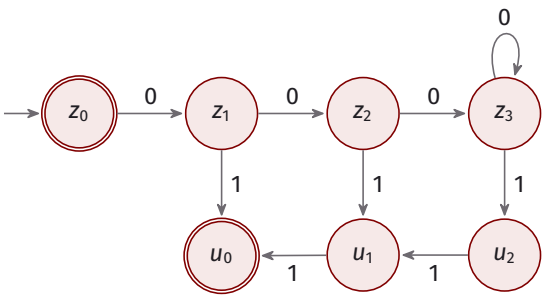
Tres semanas, una sola clase

- Semana 1: los lenguajes que reconoce algún **DFA**.
- Semana 2: un **NFA** no reconoce nada más.
- Semana 3: las **expresiones regulares** describen lo mismo.
- Unión, intersección, complemento, concatenación, estrella: no sacan de la clase.

¿Entonces todo lenguaje razonable es regular? No.

> **La forma de un resultado de imposibilidad**
>
> No decimos “nadie ha encontrado un DFA”. Decimos “**ningún** DFA lo reconoce, tenga los estados que tenga”. Hay que razonar sobre todos los autómatas a la vez.

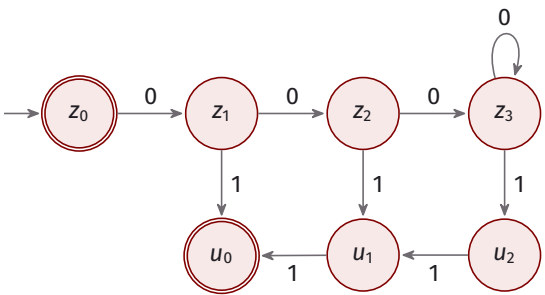
Un intento para $\{0^n 1^n\}$



Las flechas que no se ven van a un estado trampa.

- z_k: van k ceros. u_k: faltan k unos.
- En algún momento hay que dejar de agregar estados: z_3 es “3 o más”.

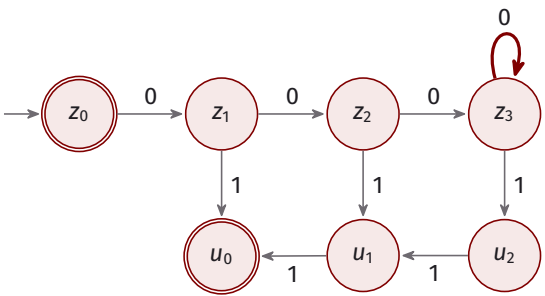
Un intento para $\{0^n 1^n\}$



Las flechas que no se ven van a un estado trampa.

- z_k: van k ceros. u_k: faltan k unos.
- En algún momento hay que dejar de agregar estados: z_3 es "3 o más".
- Acierta con ε, `01`, `0011`, `000111`.

Un intento para $\{0^n 1^n\}$



Las flechas que no se ven van a un estado trampa.

- z_k: van k ceros. u_k: faltan k unos.
- En algún momento hay que dejar de agregar estados: z_3 es "3 o más".
- Acierta con ε, `01`, `0011`, `000111`.
- Y después se equivoca: en z_3 ya no sabe cuántos ceros leyó.

El intento falla, y eso no demuestra nada

terminal

```
$ build/run_dfa machines/count_attempt.dfa 00000111 00001111
w = 00000111
  z0 -0-> z1 -0-> z2 -0-> z3 -0-> z3 -0-> z3 -1-> u2 -1-> u1 -1-> u0
  termina en u0, que está en F: acepta
w = 00001111
  z0 -0-> z1 -0-> z2 -0-> z3 -0-> z3 -1-> u2 -1-> u1 -1-> u0 -1-> x
  termina en x, que no está en F: rechaza
```

Acepta $0^5 1^3$ y rechaza $0^4 1^4$. Contar hasta 1000 solo aleja el problema.

Error común

"No se me ocurre un DFA" no demuestra nada. Hay que empezar con "sea *M* un DFA **cualquiera**".

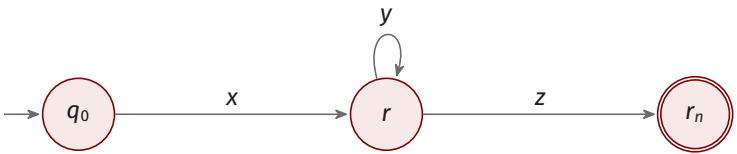
SECCIÓN 02

Un estado que se repite

02

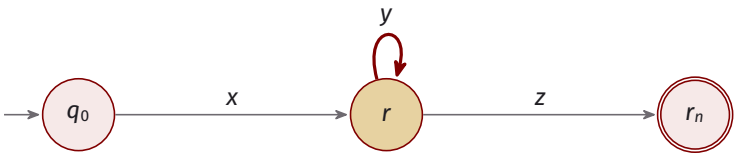
El principio del palomar

- Un DFA con p estados lee una cadena de p símbolos o más.
- Pasa por $p + 1$ estados o más, contando el inicial: **alguno se repite**.



El principio del palomar

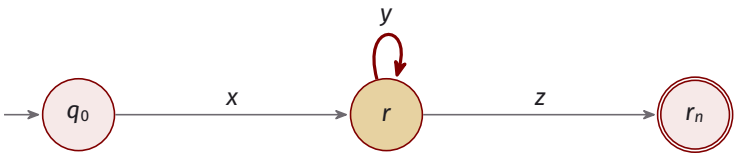
- Un DFA con p estados lee una cadena de p símbolos o más.
- Pasa por $p + 1$ estados o más, contando el inicial: **alguno se repite**.



- Entre las dos visitas a r leyó un tramo y y volvió al mismo lugar.

El principio del palomar

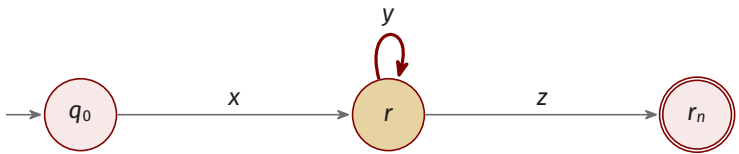
- Un DFA con p estados lee una cadena de p símbolos o más.
- Pasa por $p + 1$ estados o más, contando el inicial: **alguno se repite**.



- Entre las dos visitas a r leyó un tramo y y volvió al mismo lugar.
- El estado es todo lo que recuerda: no sabe si dio la vuelta cero, una o cien veces.

El principio del palomar

- Un DFA con p estados lee una cadena de p símbolos o más.
- Pasa por $p + 1$ estados o más, contando el inicial: **alguno se repite**.



- Entre las dos visitas a r leyó un tramo y y volvió al mismo lugar.
- El estado es todo lo que recuerda: no sabe si dio la vuelta cero, una o cien veces.

Idea clave

El tramo leído entre las dos visitas se puede quitar o repetir sin que cambie el estado final.

El ciclo en un autómata de verdad

terminal

```
$ build/pump machines/multiple_of_3.dfa 10101
p = |Q| = 3, w = 10101, |w| = 5
  r0 -1-> r1 -0-> r2 -1-> r2 -0-> r1 -1-> r0
primer estado que se repite: r2, después de leer 2 y 3 símbolos
x = 10, y = 1, z = 01    |y| = 1 > 0, |xy| = 3 ≤ 3
  i = 0: 1001 termina en r0, acepta
  i = 1: 10101 termina en r0, acepta
  i = 2: 101101 termina en r0, acepta
  i = 3: 1011101 termina en r0, acepta
  i = 4: 10111101 termina en r0, acepta
todas terminan igual que w
```

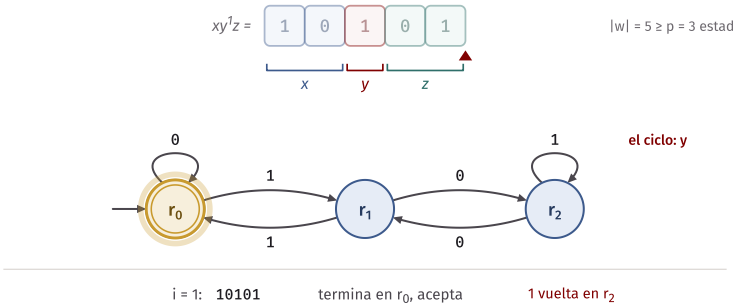
Múltiplos de 3 en binario (semana 1): $p = 3$. Las cadenas son 9, 21, 45, 93 y 189: todos múltiplos de 3.

El mismo recorrido, con más o menos vueltas



El autómata tiene 3 estados y `10101` tiene 5 símbolos. `pump` la corta en x = `10`, y = `1`, z = `01`.

El mismo recorrido, con más o menos vueltas

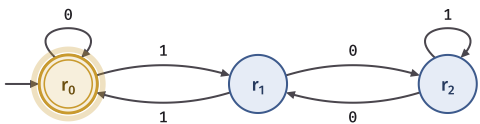


i = 1, la cadena original: pasa por r_2 después de leer 10, y el 1 siguiente la deja en r_2. Ese 1 es y.

El mismo recorrido, con más o menos vueltas



|w| = 5 ≥ p = 3 estados

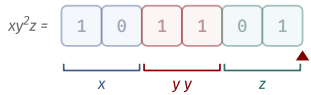


el ciclo: y

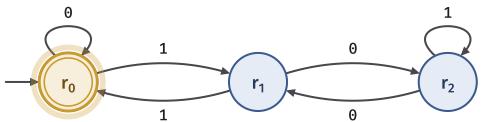
i = 1:	10101	termina en r_0, acepta	1 vuelta en r_2
i = 0:	1001	termina en r_0, acepta	0 vueltas en r_2

i = 0, sin y: llega a r_2 con x y sigue de frente con z. No da ninguna vuelta.

El mismo recorrido, con más o menos vueltas



|w| = 5 ≥ p = 3 estados



el ciclo: y

i = 1:	`10101`	termina en r_0, acepta	1 vuelta en r_2
i = 0:	`1001`	termina en r_0, acepta	0 vueltas en r_2
i = 2:	`101101`	termina en r_0, acepta	2 vueltas en r_2

i = 2, con y dos veces: da dos vueltas en r_2 y después sigue con z, igual que antes.

Las tres terminan en r_0. El estado no guarda cuántas vueltas dio la máquina.

El mismo ciclo, en el intento fallido

terminal

```
$ build/pump machines/count_attempt.dfa 00000000111
p = |Q| = 8, w = 00000000111, |w| = 11
  z0 -0-> z1 -0-> z2 -0-> z3 -0-> z3 -0-> z3 -0-> z3 -0-> z3 -0-> z3 -1-> u2 -1-> u1 -1-> u0
primer estado que se repite: z3, después de leer 3 y 4 símbolos
x = 000, y = 0, z = 0000111    |y| = 1 > 0, |xy| = 4 ≤ 8
  i = 0: 0000000111 termina en u0, acepta
  i = 1: 00000000111 termina en u0, acepta
  i = 2: 000000000111 termina en u0, acepta
  i = 3: 0000000000111 termina en u0, acepta
  i = 4: 00000000000111 termina en u0, acepta
todas terminan igual que w
```

- *y* es un cero: repetirlo cambia los ceros y no los unos.

El mismo ciclo, en el intento fallido

terminal

```
$ build/pump machines/count_attempt.dfa 00000000111
p = |Q| = 8, w = 00000000111, |w| = 11
  z0 -0-> z1 -0-> z2 -0-> z3 -0-> z3 -0-> z3 -0-> z3 -0-> z3 -0-> z3 -1-> u2 -1-> u1 -1-> u0
primer estado que se repite: z3, después de leer 3 y 4 símbolos
x = 000, y = 0, z = 0000111    |y| = 1 > 0, |xy| = 4 ≤ 8
  i = 0: 0000000111 termina en u0, acepta
  i = 1: 00000000111 termina en u0, acepta
  i = 2: 000000000111 termina en u0, acepta
  i = 3: 0000000000111 termina en u0, acepta
  i = 4: 00000000000111 termina en u0, acepta
todas terminan igual que w
```

- *y* es un cero: repetirlo cambia los ceros y no los unos.
- No es un accidente de este diseño. **Cualquier** DFA, con una cadena 0^n1^n larga, repite un estado mientras todavía lee ceros.

SECCIÓN 03

El lema de bombeo

03

El lema de bombeo

Lema de bombeo ***(pumping lemma)***

Si A es regular, existe un número $p \geq 1$, la **longitud de bombeo**, tal que toda cadena $s \in A$ con $|s| \geq p$ se puede dividir en $s = xyz$ con:

1. $xy^i z \in A$ para todo $i \geq 0$;
2. $|y| > 0$;
3. $|xy| \leq p$.

El lema de bombeo

Lema de bombeo ***(pumping lemma)***

Si A es regular, existe un número $p \geq 1$, la **longitud de bombeo**, tal que toda cadena $s \in A$ con $|s| \geq p$ se puede dividir en $s = xyz$ con:

1. $xy^i z \in A$ para todo $i \geq 0$;
2. $|y| > 0$;
3. $|xy| \leq p$.

- (1) es el bombeo. Con $i = 0$ sale xz, la cadena **sin** y.

El lema de bombeo

Lema de bombeo ***(pumping lemma)***

Si A es regular, existe un número $p \geq 1$, la **longitud de bombeo**, tal que toda cadena $s \in A$ con $|s| \geq p$ se puede dividir en $s = xyz$ con:

1. $xy^iz \in A$ para todo $i \geq 0$;
2. $|y| > 0$;
3. $|xy| \leq p$.

- (1) es el bombeo. Con $i = 0$ sale xz, la cadena **sin** y.
- (2) sin ella el lema no diría nada: con $y = \varepsilon$ todas las xy^iz son s.

El lema de bombeo

Lema de bombeo ***(pumping lemma)***

Si A es regular, existe un número $p \geq 1$, la **longitud de bombeo**, tal que toda cadena $s \in A$ con $|s| \geq p$ se puede dividir en $s = xyz$ con:

1. $xy^iz \in A$ para todo $i \geq 0$;
2. $|y| > 0$;
3. $|xy| \leq p$.

- (1) es el bombeo. Con $i = 0$ sale xz, la cadena **sin** y.
- (2) sin ella el lema no diría nada: con $y = \varepsilon$ todas las xy^iz son s.
- (3) dice **dónde** está y: dentro de los primeros p símbolos.

El lema de bombeo

Lema de bombeo ***(pumping lemma)***

Si A es regular, existe un número $p \geq 1$, la **longitud de bombeo**, tal que toda cadena $s \in A$ con $|s| \geq p$ se puede dividir en $s = xyz$ con:

1. $xy^iz \in A$ para todo $i \geq 0$;
2. $|y| > 0$;
3. $|xy| \leq p$.

- (1) es el bombeo. Con $i = 0$ sale xz, la cadena **sin** y.
- (2) sin ella el lema no diría nada: con $y = \varepsilon$ todas las xy^iz son s.
- (3) dice **dónde** está y: dentro de los primeros p símbolos.
- De las cadenas más cortas que p no dice nada: un lenguaje finito lo cumple.

Demostración: un estado se repite pronto

Sea M un DFA con $L(M) = A$. Tomamos $p = |Q|$.

- Sea $s = s_1 \cdots s_n \in A$ con $n \geq p$, y $r_0, r_1, \ldots, r_n$ la ejecución. $r_n \in F$.

Demostración: un estado se repite pronto

Sea M un DFA con $L(M) = A$. Tomamos $p = |Q|$.

- Sea $s = s_1 \cdots s_n \in A$ con $n \geq p$, y $r_0, r_1, \ldots, r_n$ la ejecución. $r_n \in F$.
- Miramos solo los primeros $p + 1$ estados: $r_0, r_1, \ldots, r_p$.

Demostración: un estado se repite pronto

Sea M un DFA con $L(M) = A$. Tomamos $p = |Q|$.

- Sea $s = s_1 \cdots s_n \in A$ con $n \geq p$, y $r_0, r_1, \ldots, r_n$ la ejecución. $r_n \in F$.
- Miramos solo los primeros $p + 1$ estados: $r_0, r_1, \ldots, r_p$.
- Hay p estados distintos: por el palomar, $r_j = r_l$ con $0 \leq j < l \leq p$.

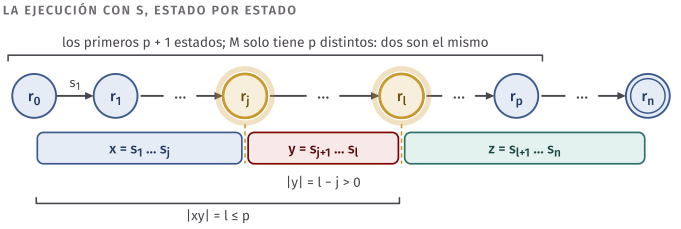
Demostración: un estado se repite pronto

Sea M un DFA con $L(M) = A$. Tomamos $p = |Q|$.

- Sea $s = s_1 \cdots s_n \in A$ con $n \geq p$, y $r_0, r_1, \ldots, r_n$ la ejecución. $r_n \in F$.
- Miramos solo los primeros $p + 1$ estados: $r_0, r_1, \ldots, r_p$.
- Hay p estados distintos: por el palomar, $r_j = r_l$ con $0 \leq j < l \leq p$.

Cortamos s en esas dos visitas: $x = s_1 \cdots s_j$, $y = s_{j+1} \cdots s_l$, $z = s_{l+1} \cdots s_n$.

Demostración: la división



Cada flecha punteada es un tramo de varios símbolos.
Con xy^iz, la máquina da i vueltas y termina en el mismo r_n, que está en F.

Demostración: las tres condiciones

Sea $r = r_j = r_l$. Entonces

$$\delta^*(q_0, x) = r, \qquad \delta^*(r, y) = r, \qquad \delta^*(r, z) = r_n \in F.$$

Demostración: las tres condiciones

Sea $r = r_j = r_l$. Entonces

$$\delta^*(q_0, x) = r, \qquad \delta^*(r, y) = r, \qquad \delta^*(r, z) = r_n \in F.$$

- **(2)** $|y| = l - j > 0$, porque $j < l$.

Demostración: las tres condiciones

Sea $r = r_j = r_l$. Entonces

$$\delta^*(q_0, x) = r, \qquad \delta^*(r, y) = r, \qquad \delta^*(r, z) = r_n \in F.$$

- **(2)** $|y| = l - j > 0$, porque $j < l$.
- **(3)** $|xy| = l \leq p$.

Demostración: las tres condiciones

Sea $r = r_j = r_l$. Entonces

$$\delta^*(q_0, x) = r, \qquad \delta^*(r, y) = r, \qquad \delta^*(r, z) = r_n \in F.$$

- **(2)** $|y| = l - j > 0$, porque $j < l$.
- **(3)** $|xy| = l \leq p$.
- **(1)** Por inducción sobre i: $\delta^*(r, y^i) = r$. Y entonces

$$\delta^*(q_0, xy^i z) = \delta^*(\delta^*(r, y^i), z) = \delta^*(r, z) = r_n \in F. \qquad \blacksquare$$

Demostración: las tres condiciones

Sea $r = r_j = r_l$. Entonces

$$\delta^*(q_0, x) = r, \qquad \delta^*(r, y) = r, \qquad \delta^*(r, z) = r_n \in F.$$

- **(2)** $|y| = l - j > 0$, porque $j < l$.
- **(3)** $|xy| = l \leq p$.
- **(1)** Por inducción sobre i: $\delta^*(r, y^i) = r$. Y entonces

$$\delta^*(q_0, xy^iz) = \delta^*(\delta^*(r, y^i), z) = \delta^*(r, z) = r_n \in F. \qquad \blacksquare$$

La división la decide un autómata que no tenemos: el lema solo promete que **alguna** funciona.

SECCIÓN 04

El lema como un juego

04

Usarlo al revés

Si A es regular:

- existe p tal que
- para toda $s \in A$, $|s| \geq p$,
- existe una división $s = xyz$ con $|y| > 0$, $|xy| \leq p$, tal que
- para todo $i \geq 0$: $xy^i z \in A$.

Usarlo al revés

Si A es regular:

- existe p tal que
- para toda $s \in A$, $|s| \geq p$,
- existe una división $s = xyz$ con $|y| > 0$, $|xy| \leq p$, tal que
- para todo $i \geq 0$: $xy^i z \in A$.

Para mostrar que no lo es:

- para todo p
- existe $s \in A$, $|s| \geq p$, tal que
- para toda división $s = xyz$ con $|y| > 0$, $|xy| \leq p$,
- existe $i \geq 0$: $xy^i z \notin A$.

Usarlo al revés

Si A es regular:

- existe p tal que
- para toda $s \in A$, $|s| \geq p$,
- existe una división $s = xyz$ con $|y| > 0$, $|xy| \leq p$, tal que
- para todo $i \geq 0$: $xy^i z \in A$.

Para mostrar que no lo es:

- para todo p
- existe $s \in A$, $|s| \geq p$, tal que
- para toda división $s = xyz$ con $|y| > 0$, $|xy| \leq p$,
- existe $i \geq 0$: $xy^i z \notin A$.

Contrarrecíproca

Si A no tiene la propiedad de bombeo, A no es regular. Cada “existe” se volvió “para todo”, y al revés.

Un juego contra un adversario

Jugada	Quién	Qué elige
1	El adversario	Un número $p \geq 1$

Un juego contra un adversario

Jugada	Quién	Qué elige
1	El adversario	Un número $p \geq 1$
2	Nosotros	Una cadena $s \in A$ con $\|s\| \geq p$. Depende de p

Un juego contra un adversario

Jugada	Quién	Qué elige
1	El adversario	Un número $p \geq 1$
2	Nosotros	Una cadena $s \in A$ con $\|s\| \geq p$. Depende de p
3	El adversario	Una división $s = xyz$ con $\|y\| > 0$ y $\|xy\| \leq p$

Un juego contra un adversario

Jugada	Quién	Qué elige
1	El adversario	Un número $p \geq 1$
2	Nosotros	Una cadena $s \in A$ con $\|s\| \geq p$. Depende de p
3	El adversario	Una división $s = xyz$ con $\|y\| > 0$ y $\|xy\| \leq p$
4	Nosotros	Un exponente $i \geq 0$. Ganamos si $xy^i z \notin A$

Un juego contra un adversario

Jugada	Quién	Qué elige
1	El adversario	Un número $p \geq 1$
2	Nosotros	Una cadena $s \in A$ con $\lvert s\rvert \geq p$. Depende de p
3	El adversario	Una división $s = xyz$ con $\lvert y\rvert > 0$ y $\lvert xy\rvert \leq p$
4	Nosotros	Un exponente $i \geq 0$. Ganamos si $xy^iz \notin A$

- **No conocemos** p: s es una fórmula con p, como 0^p1^p.

Un juego contra un adversario

Jugada	Quién	Qué elige
1	El adversario	Un número $p \geq 1$
2	Nosotros	Una cadena $s \in A$ con $\lvert s\rvert \geq p$. Depende de p
3	El adversario	Una división $s = xyz$ con $\lvert y\rvert > 0$ y $\lvert xy\rvert \leq p$
4	Nosotros	Un exponente $i \geq 0$. Ganamos si $xy^iz \notin A$

- **No conocemos** p: s es una fórmula con p, como 0^p1^p.
- **No elegimos la división**: el argumento cubre *todas* las permitidas.

Un juego contra un adversario

Jugada	Quién	Qué elige
1	El adversario	Un número $p \geq 1$
2	Nosotros	Una cadena $s \in A$ con $\lvert s\rvert \geq p$. Depende de p
3	El adversario	Una división $s = xyz$ con $\lvert y\rvert > 0$ y $\lvert xy\rvert \leq p$
4	Nosotros	Un exponente $i \geq 0$. Ganamos si $xy^i z \notin A$

- **No conocemos** p: s es una fórmula con p, como $0^p 1^p$.
- **No elegimos la división**: el argumento cubre *todas* las permitidas.
- Una buena s usa $\lvert xy\rvert \leq p$ para arrinconar a y.

$\{0^n 1^n\}$ no es regular

Supongamos que $B = \{0^n 1^n : n \geq 0\}$ es regular.

$\{0^n 1^n\}$ no es regular

Supongamos que $B = \{0^n 1^n : n \geq 0\}$ es regular.

1. **El adversario elige p.** No sabemos cuánto vale.

$\{0^n 1^n\}$ no es regular

Supongamos que $B = \{0^n 1^n : n \geq 0\}$ es regular.

1. **El adversario elige** p**.** No sabemos cuánto vale.
2. **Elegimos** $s = 0^p 1^p$. Está en B y $|s| = 2p \geq p$.

$\{0^n 1^n\}$ no es regular

Supongamos que $B = \{0^n 1^n : n \geq 0\}$ es regular.

1. **El adversario elige** p**.** No sabemos cuánto vale.
2. **Elegimos** $s = 0^p 1^p$. Está en B y $|s| = 2p \geq p$.
3. **El adversario elige** $s = xyz$ con $|y| > 0$ y $|xy| \leq p$.
   Los primeros p símbolos son ceros: $y = 0^k$ con $k \geq 1$, **sea cual sea la división**.

$\{0^n 1^n\}$ no es regular

Supongamos que $B = \{0^n 1^n : n \geq 0\}$ es regular.

1. **El adversario elige** p**.** No sabemos cuánto vale.
2. **Elegimos** $s = 0^p 1^p$. Está en B y $|s| = 2p \geq p$.
3. **El adversario elige** $s = xyz$ con $|y| > 0$ y $|xy| \leq p$.
   Los primeros p símbolos son ceros: $y = 0^k$ con $k \geq 1$, **sea cual sea la división**.
4. **Elegimos** $i = 2$: $xy^2z = 0^{p+k}1^p$. Más ceros que unos: no está en B.

$\{0^n 1^n\}$ no es regular

Supongamos que $B = \{0^n 1^n : n \geq 0\}$ es regular.

1. **El adversario elige** p**.** No sabemos cuánto vale.
2. **Elegimos** $s = 0^p 1^p$. Está en B y $|s| = 2p \geq p$.
3. **El adversario elige** $s = xyz$ con $|y| > 0$ y $|xy| \leq p$.
   Los primeros p símbolos son ceros: $y = 0^k$ con $k \geq 1$, **sea cual sea la división**.
4. **Elegimos** $i = 2$: $xy^2z = 0^{p+k}1^p$. Más ceros que unos: no está en B.

Pero el lema dice que $xy^2z \in B$. Contradicción. ■

$\{0^n 1^n\}$ no es regular

Supongamos que $B = \{0^n 1^n : n \geq 0\}$ es regular.

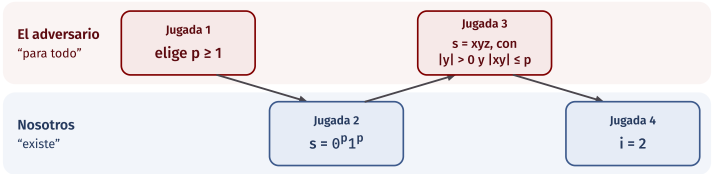
1. **El adversario elige** p. No sabemos cuánto vale.
2. **Elegimos** $s = 0^p 1^p$. Está en B y $|s| = 2p \geq p$.
3. **El adversario elige** $s = xyz$ con $|y| > 0$ y $|xy| \leq p$.
   Los primeros p símbolos son ceros: $y = 0^k$ con $k \geq 1$, **sea cual sea la división**.
4. **Elegimos** $i = 2$: $xy^2z = 0^{p+k}1^p$. Más ceros que unos: no está en B.

Pero el lema dice que $xy^2z \in B$. Contradicción. ■

La condición 3 hizo el trabajo

Sin $|xy| \leq p$, el adversario podría poner unos en y, y habría que tratar más casos.

La partida, en un dibujo



Sea cual sea la división permitida, xy^2z tiene k ceros de más: ganamos la partida para todo p.

La partida completa, con $p = 3$

terminal

```
$ build/pumping_game 0n1n 3
L = { 0^n 1^n : n ≥ 0 }
el adversario elige p = 3
nosotros elegimos s = 000111, con |s| = 6 y s ∈ L
nuestro exponente: i = 2
divisiones posibles con |xy| ≤ 3 y |y| > 0: 6
  x   y    z      x y^i z
  ε   0    00111  0000111    ∉ L
  ε   00   0111   00000111   ∉ L
  ε   000  111    000000111  ∉ L
  0   0    0111   0000111    ∉ L
  0   00   111    00000111   ∉ L
  00  0    111    0000111    ∉ L
en las 6 divisiones la cadena bombeada sale de L
```

Con $|s| \geq p$ hay $p(p+1)/2$ divisiones permitidas. Jugar con un p no demuestra nada: sirve para encontrar errores.

Dos maneras de perder antes de empezar

Error común: elegir la división

"Sea $x = 0^{p-1}$, $y = 0$, $z = 1^p$. Entonces $xy^2z \notin A$." Eso muestra que **una** división falla. El lema promete que **alguna** funciona, no que todas.

Dos maneras de perder antes de empezar

Error común: elegir la división

"Sea $x = 0^{p-1}$, $y = 0$, $z = 1^p$. Entonces $xy^2z \notin A$." Eso muestra que **una** división falla. El lema promete que **alguna** funciona, no que todas.

Error común: una cadena que no depende de p

"Sea $s = 000111$." Si el adversario dice $p = 7$, esa cadena es más corta que p y el lema no dice nada sobre ella.

Dos maneras de perder antes de empezar

Error común: elegir la división

"Sea $x = 0^{p-1}$, $y = 0$, $z = 1^p$. Entonces $xy^2z \notin A$." Eso muestra que **una** división falla. El lema promete que **alguna** funciona, no que todas.

Error común: una cadena que no depende de p

"Sea $s = 000111$." Si el adversario dice $p = 7$, esa cadena es más corta que p y el lema no dice nada sobre ella.

La división es la jugada del adversario. La cadena es la nuestra, y la damos **después** de oír p.

Igual número de ceros y unos: importa la cadena

$E = \{w : w \text{ tiene tantos ceros como unos}\}$. Con $s = 0^p 1^p$, la demostración anterior sirve tal cual.

Con $s = (01)^p$, que también está en E, perdemos:

terminal

```
$ build/pumping_game iguales-mal 3
  x   y    z      x y^i z
  ε   0    10101  0010101    ∉ L
  ε   01   0101   01010101   ∈ L: el adversario resiste
  ε   010  101    010010101  ∉ L
  0   1    0101   0110101    ∉ L
  0   10   101    01010101   ∈ L: el adversario resiste
  01  0    101    0100101    ∉ L
el adversario tiene 2 divisiones que resisten: con esta jugada no se demuestra nada
```

- El adversario elige $y = 01$: todas las $xy^iz = (01)^{p-1+i}$ están en E.

Igual número de ceros y unos: importa la cadena

$E = \{w : w \text{ tiene tantos ceros como unos}\}$. Con $s = 0^p 1^p$, la demostración anterior sirve tal cual.

Con $s = (01)^p$, que también está en E, perdemos:

terminal

```
$ build/pumping_game iguales-mal 3
  x   y    z      x y^i z
  ε   0    10101  0010101    ∉ L
  ε   01   0101   01010101   ∈ L: el adversario resiste
  ε   010  101    010010101  ∉ L
  0   1    0101   0110101    ∉ L
  0   10   101    01010101   ∈ L: el adversario resiste
  01  0    101    0100101    ∉ L
el adversario tiene 2 divisiones que resisten: con esta jugada no se demuestra nada
```

- El adversario elige $y = 01$: todas las $xy^iz = (01)^{p-1+i}$ están en E.
- Una partida perdida no dice que E sea regular. Dice que hay que elegir otra cadena.

$\{ww\}$ no es regular

$D = \{ww : w \in \{0,1\}^*\}$. Supongamos que es regular.

1. El adversario elige p.

$\{ww\}$ no es regular

$D = \{ww : w \in \{0,1\}^*\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = 0^p 1\, 0^p 1$. Está en D, con $w = 0^p 1$.

$\{ww\}$ no es regular

$D = \{ww : w \in \{0,1\}^*\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = 0^p 1\, 0^p 1$. Está en D, con $w = 0^p 1$.
3. El adversario divide: $y = 0^k$, $1 \leq k \leq p$, dentro del primer bloque de ceros.

$\{ww\}$ no es regular

$D = \{ww : w \in \{0,1\}^*\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = 0^p 1\, 0^p 1$. Está en D, con $w = 0^p 1$.
3. El adversario divide: $y = 0^k$, $1 \leq k \leq p$, dentro del primer bloque de ceros.
4. Elegimos $i = 2$: $xy^2z = 0^{p+k} 1\, 0^p 1$.

$\{ww\}$ no es regular

$D = \{ww : w \in \{0,1\}^*\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = 0^p 1\, 0^p 1$. Está en D, con $w = 0^p 1$.
3. El adversario divide: $y = 0^k$, $1 \leq k \leq p$, dentro del primer bloque de ceros.
4. Elegimos $i = 2$: $xy^2z = 0^{p+k} 1\, 0^p 1$.

¿Puede ser uu? Termina en 1 y tiene dos unos: u tiene un solo 1, al final. Entonces $uu = 0^m 1\, 0^m 1$, con $m = p + k$ y $m = p$. Imposible, porque $k \geq 1$. ■

$\{ww\}$ no es regular

$D = \{ww : w \in \{0,1\}^*\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = 0^p 1\, 0^p 1$. Está en D, con $w = 0^p 1$.
3. El adversario divide: $y = 0^k$, $1 \leq k \leq p$, dentro del primer bloque de ceros.
4. Elegimos $i = 2$: $xy^2z = 0^{p+k} 1\, 0^p 1$.

¿Puede ser uu? Termina en `1` y tiene dos unos: u tiene un solo `1`, al final. Entonces $uu = 0^m 1\, 0^m 1$, con $m = p + k$ y $m = p$. Imposible, porque $k \geq 1$. ■

En la práctica

El patrón `^(.*)\1$` describe D. Un motor que acepte `\1` no puede ser un autómata finito: es la deuda de la semana 3.

Los cuadrados: un argumento aritmético

$C = \{\mathtt{o}^{n^2} : n \geq 0\}$, sobre el alfabeto $\{\mathtt{o}\}$. Supongamos que es regular.

1. El adversario elige p.

Los cuadrados: un argumento aritmético

$C = \{\circ^{n^2} : n \geq 0\}$, sobre el alfabeto $\{\circ\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = \circ^{p^2}$. Está en C y $|s| = p^2 \geq p$.

Los cuadrados: un argumento aritmético

$C = \{\circ^{n^2} : n \geq 0\}$, sobre el alfabeto $\{\circ\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = \circ^{p^2}$. Está en C y $|s| = p^2 \geq p$.
3. El adversario divide. Sea $k = |y|$: $1 \leq k \leq p$.

Los cuadrados: un argumento aritmético

$C = \{\circ^{n^2} : n \geq 0\}$, sobre el alfabeto $\{\circ\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = \circ^{p^2}$. Está en C y $|s| = p^2 \geq p$.
3. El adversario divide. Sea $k = |y|$: $1 \leq k \leq p$.
4. Elegimos $i = 2$. La longitud de xy^2z es $p^2 + k$, y

$$p^2 \ < \ p^2 + k \ \leq \ p^2 + p \ < \ (p+1)^2.$$

Los cuadrados: un argumento aritmético

$C = \{\circ^{n^2} : n \geq 0\}$, sobre el alfabeto $\{\circ\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = \circ^{p^2}$. Está en C y $|s| = p^2 \geq p$.
3. El adversario divide. Sea $k = |y|$: $1 \leq k \leq p$.
4. Elegimos $i = 2$. La longitud de xy^2z es $p^2 + k$, y

$$p^2 \;<\; p^2 + k \;\leq\; p^2 + p \;<\; (p+1)^2.$$

Está estrictamente entre dos cuadrados consecutivos: no es un cuadrado. ■

Los cuadrados: un argumento aritmético

$C = \{\circ^{n^2} : n \geq 0\}$, sobre el alfabeto $\{\circ\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = \circ^{p^2}$. Está en C y $|s| = p^2 \geq p$.
3. El adversario divide. Sea $k = |y|$: $1 \leq k \leq p$.
4. Elegimos $i = 2$. La longitud de xy^2z es $p^2 + k$, y

$$p^2 \;<\; p^2 + k \;\leq\; p^2 + p \;<\; (p+1)^2.$$

Está estrictamente entre dos cuadrados consecutivos: no es un cuadrado. ■

El salto de p^2 al cuadrado siguiente es $2p + 1$, y el adversario solo puede sumar entre 1 y p.

Más ceros que unos: bombear hacia abajo

$G = \{0^i 1^j : i > j\}$. Supongamos que es regular.

1. El adversario elige p.

Más ceros que unos: bombear hacia abajo

$G = \{0^i 1^j : i > j\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = 0^{p+1}1^p$: un solo cero de ventaja.

Más ceros que unos: bombear hacia abajo

$G = \{0^i 1^j : i > j\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = 0^{p+1}1^p$: un solo cero de ventaja.
3. El adversario divide: $y = 0^k$, $1 \leq k \leq p$.

Más ceros que unos: bombear hacia abajo

$G = \{0^i 1^j : i > j\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = 0^{p+1}1^p$: un solo cero de ventaja.
3. El adversario divide: $y = 0^k$, $1 \leq k \leq p$.
4. Con $i = 2$ hay *más* ceros: la cadena sigue en G. Bombear hacia arriba no sirve.

Más ceros que unos: bombear hacia abajo

$G = \{0^i 1^j : i > j\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = 0^{p+1}1^p$: un solo cero de ventaja.
3. El adversario divide: $y = 0^k$, $1 \leq k \leq p$.
4. Con $i = 2$ hay *más* ceros: la cadena sigue en G. Bombear hacia arriba no sirve.
5. Elegimos $i = 0$: $xz = 0^{p+1-k}1^p$, con $p + 1 - k \leq p$ ceros. No está en G. ■

Más ceros que unos: bombear hacia abajo

$G = \{0^i 1^j : i > j\}$. Supongamos que es regular.

1. El adversario elige p.
2. Elegimos $s = 0^{p+1}1^p$: un solo cero de ventaja.
3. El adversario divide: $y = 0^k$, $1 \leq k \leq p$.
4. Con $i = 2$ hay *más* ceros: la cadena sigue en G. Bombear hacia arriba no sirve.
5. Elegimos $i = 0$: $xz = 0^{p+1-k}1^p$, con $p + 1 - k \leq p$ ceros. No está en G. ■

Idea clave

$i = 0$ es tan legítimo como $i = 2$. La cadena se elige en el borde de la condición, para que el menor cambio la saque.

Combinar con las propiedades de cierre

Otra demostración de que E no es regular.

- Si E fuera regular, $E \cap \mathtt{0}^*\mathtt{1}^*$ también: los regulares son cerrados bajo intersección.

Combinar con las propiedades de cierre

Otra demostración de que E no es regular.

- Si E fuera regular, $E \cap \mathtt{0}^*\mathtt{1}^*$ también: los regulares son cerrados bajo intersección.
- Pero $E \cap \mathtt{0}^*\mathtt{1}^* = \{\mathtt{0}^n\mathtt{1}^n\}$, que no es regular. Contradicción.

Combinar con las propiedades de cierre

Otra demostración de que E no es regular.

- Si E fuera regular, $E \cap \mathtt{0}^*\mathtt{1}^*$ también: los regulares son cerrados bajo intersección.
- Pero $E \cap \mathtt{0}^*\mathtt{1}^* = \{\mathtt{0}^n\mathtt{1}^n\}$, que no es regular. Contradicción.
- Con el complemento: “distinta cantidad de ceros y unos” tampoco es regular.

Combinar con las propiedades de cierre

Otra demostración de que E no es regular.

- Si E fuera regular, $E \cap \mathtt{0}^*\mathtt{1}^*$ también: los regulares son cerrados bajo intersección.
- Pero $E \cap \mathtt{0}^*\mathtt{1}^* = \{\mathtt{0}^n\mathtt{1}^n\}$, que no es regular. Contradicción.
- Con el complemento: "distinta cantidad de ceros y unos" tampoco es regular.

Error común: la dirección

De "L contiene a un lenguaje no regular" no sale nada: Σ^* los contiene a todos. De "L está contenido en uno no regular" tampoco: $\emptyset$ está en todos. El lenguaje sospechoso va entre los **datos**, y los demás datos son regulares.

Lo que el lema no dice

- El lema es una implicación en un solo sentido: regular $\Rightarrow$ se puede bombear.

Lo que el lema no dice

- El lema es una implicación en un solo sentido: regular $\Rightarrow$ se puede bombear.
- El otro sentido es **falso**. Sobre $\{a, b, c\}$:

$$T = \{\, a^i b^j c^k : \text{si } i = 1 \text{ entonces } j = k \,\}$$

Lo que el lema no dice

- El lema es una implicación en un solo sentido: regular $\Rightarrow$ se puede bombear.
- El otro sentido es **falso**. Sobre $\{a, b, c\}$:

$$T = \{\, a^i b^j c^k : \text{si } i = 1 \text{ entonces } j = k \,\}$$

- T se deja bombear con $p = 2$: y es la primera letra, o aa si empieza con exactamente dos a.

Lo que el lema no dice

- El lema es una implicación en un solo sentido: regular $\Rightarrow$ se puede bombear.
- El otro sentido es **falso**. Sobre $\{a, b, c\}$:

$$T = \{\, a^i b^j c^k : \text{si } i = 1 \text{ entonces } j = k \,\}$$

- T se deja bombear con $p = 2$: y es la primera letra, o aa si empieza con exactamente dos a.
- Y T no es regular: $T \cap ab^*c^* = \{ab^n c^n\}$.

Lo que el lema no dice

- El lema es una implicación en un solo sentido: regular $\Rightarrow$ se puede bombear.
- El otro sentido es **falso**. Sobre $\{a, b, c\}$:

$$T = \{\, a^i b^j c^k : \text{si } i = 1 \text{ entonces } j = k \,\}$$

- T se deja bombear con $p = 2$: y es la primera letra, o aa si empieza con exactamente dos a.
- Y T no es regular: $T \cap ab^*c^* = \{ab^n c^n\}$.

Error común

"Encontré una división que se puede bombear, así que es regular." Para demostrar que un lenguaje **es** regular se da un DFA, un NFA o una expresión regular.

SECCIÓN 05

Cadenas distinguibles

05



Dos cadenas con futuros distintos

Cadenas distinguibles *(distinguishable strings)*

x e y son **distinguibles** respecto de L si existe un sufijo z tal que **exactamente una** de xz, yz está en L. Si no existe, son indistinguibles: $x \equiv_L y$.

Respecto de $B = \{0^n 1^n\}$:

Dos cadenas con futuros distintos

Cadenas distinguibles *(distinguishable strings)*

x e y son **distinguibles** respecto de L si existe un sufijo z tal que **exactamente una** de xz, yz está en L. Si no existe, son indistinguibles: $x \equiv_L y$.

Respecto de $B = \{0^n 1^n\}$:

- `000` y `00000` son distinguibles: $z = 111$.

Dos cadenas con futuros distintos

Cadenas distinguibles *(distinguishable strings)*

x e y son **distinguibles** respecto de L si existe un sufijo z tal que **exactamente una** de xz, yz está en L. Si no existe, son indistinguibles: $x \equiv_L y$.

Respecto de $B = \{0^n 1^n\}$:

- `000` y `00000` son distinguibles: $z = 111$.
- `01` y `0011` son indistinguibles: las dos están en B, y con $z \neq \varepsilon$ ninguna.

Dos cadenas con futuros distintos

Cadenas distinguibles *(distinguishable strings)*

x e y son **distinguibles** respecto de L si existe un sufijo z tal que **exactamente una** de xz, yz está en L. Si no existe, son indistinguibles: $x \equiv_L y$.

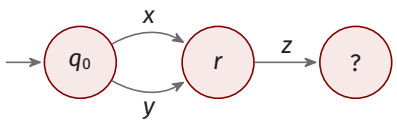
Respecto de $B = \{0^n 1^n\}$:

- `000` y `00000` son distinguibles: $z = 111$.
- `01` y `0011` son indistinguibles: las dos están en B, y con $z \neq \varepsilon$ ninguna.
- x e y no tienen que estar en L. Se compara lo que pasa **después**.
- z es **el mismo** para las dos, y puede ser ε.

No pueden terminar en el mismo estado

Lema de los estados distintos

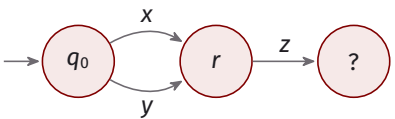
Si M reconoce L y x, y son distinguibles, entonces $\delta^*(q_0, x) \neq \delta^*(q_0, y)$.



No pueden terminar en el mismo estado

Lema de los estados distintos

Si M reconoce L y x, y son distinguibles, entonces $\delta^*(q_0, x) \neq \delta^*(q_0, y)$.

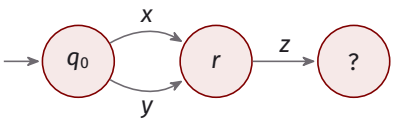


- Si x e y llegan al mismo estado r, entonces xz e yz terminan en el mismo estado, para toda z.

No pueden terminar en el mismo estado

Lema de los estados distintos

Si M reconoce L y x, y son distinguibles, entonces $\delta^*(q_0, x) \neq \delta^*(q_0, y)$.

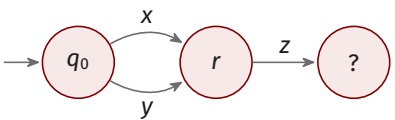


- Si x e y llegan al mismo estado r, entonces xz e yz terminan en el mismo estado, para toda z.
- M acepta las dos o rechaza las dos: ninguna z las distingue.

No pueden terminar en el mismo estado

Lema de los estados distintos

Si M reconoce L y x, y son distinguibles, entonces $\delta^*(q_0, x) \neq \delta^*(q_0, y)$.



- Si x e y llegan al mismo estado r, entonces xz e yz terminan en el mismo estado, para toda z.
- M acepta las dos o rechaza las dos: ninguna z las distingue.

Cota inferior de estados

k cadenas distinguibles **dos a dos**: al menos k estados. Infinitas: L no es regular.

$\{0^n 1^n\}$ otra vez, en tres líneas

- Sea $S = \{0^i : i \geq 0\}$, que es infinito.

$\{0^n 1^n\}$ otra vez, en tres líneas

- Sea $S = \{0^i : i \geq 0\}$, que es infinito.
- Si $i \neq j$, el sufijo $z = 1^i$ distingue a 0^i de 0^j: $0^i 1^i \in B$ y $0^j 1^i \notin B$.

$\{0^n 1^n\}$ otra vez, en tres líneas

- Sea $S = \{0^i : i \geq 0\}$, que es infinito.
- Si $i \neq j$, el sufijo $z = 1^i$ distingue a 0^i de 0^j: $0^i 1^i \in B$ y $0^j 1^i \notin B$.
- Infinitas cadenas distinguibles dos a dos: B no es regular. ■

$\{0^n 1^n\}$ otra vez, en tres líneas

- Sea $S = \{0^i : i \geq 0\}$, que es infinito.
- Si $i \neq j$, el sufijo $z = 1^i$ distingue a 0^i de 0^j: $0^i 1^i \in B$ y $0^j 1^i \notin B$.
- Infinitas cadenas distinguibles dos a dos: B no es regular. ■

Sin adversario, sin división, sin casos. Y los demás salen igual:

- G, más ceros que unos: 0^i, 0^j con $i < j$ y $z = 1^i$.
- C, los cuadrados: 0^{i^2}, 0^{j^2} con $i < j$ y $z = 0^{2i+1}$.
- T, el que resistía al bombeo: ab^i, ab^j y $z = c^i$.

$\{0^n 1^n\}$ otra vez, en tres líneas

- Sea $S = \{0^i : i \geq 0\}$, que es infinito.
- Si $i \neq j$, el sufijo $z = 1^i$ distingue a 0^i de 0^j: $0^i 1^i \in B$ y $0^j 1^i \notin B$.
- Infinitas cadenas distinguibles dos a dos: B no es regular. ■

Sin adversario, sin división, sin casos. Y los demás salen igual:

- G, más ceros que unos: 0^i, 0^j con $i < j$ y $z = 1^i$.
- C, los cuadrados: 0^{i^2}, 0^{j^2} con $i < j$ y $z = 0^{2i+1}$.
- T, el que resistía al bombeo: ab^i, ab^j y $z = c^i$.

Error común

Distinguir solo algunos pares. Hace falta un sufijo para **todo** par $i \neq j$.

La deuda de la semana 2

L_k: el k-ésimo símbolo desde el final es 1. NFA: $k+1$ estados.

Teorema

Todo DFA que reconoce L_k tiene al menos 2^k estados.

La deuda de la semana 2

L_k: el k-ésimo símbolo desde el final es 1. NFA: $k+1$ estados.

Teorema
Todo DFA que reconoce L_k tiene al menos 2^k estados.

- S: las 2^k cadenas de longitud k.

La deuda de la semana 2

L_k: el k-ésimo símbolo desde el final es 1. NFA: $k+1$ estados.

Teorema
Todo DFA que reconoce L_k tiene al menos 2^k estados.

- S: las 2^k cadenas de longitud k.
- Si $x \neq y$, difieren en alguna posición i. Digamos $x_i = 1$, $y_i = 0$.

La deuda de la semana 2

L_k: el k-ésimo símbolo desde el final es 1. NFA: $k+1$ estados.

Teorema

Todo DFA que reconoce L_k tiene al menos 2^k estados.

- S: las 2^k cadenas de longitud k.
- Si $x \neq y$, difieren en alguna posición i. Digamos $x_i = 1$, $y_i = 0$.
- Tomamos $z = 0^{i-1}$. En xz, de longitud $k + i - 1$, el k-ésimo desde el final es la posición i.

La deuda de la semana 2

L_k: el k-ésimo símbolo desde el final es 1. NFA: $k+1$ estados.

Teorema

Todo DFA que reconoce L_k tiene al menos 2^k estados.

- S: las 2^k cadenas de longitud k.
- Si $x \neq y$, difieren en alguna posición i. Digamos $x_i = 1$, $y_i = 0$.
- Tomamos $z = 0^{i-1}$. En xz, de longitud $k + i - 1$, el k-ésimo desde el final es la posición i.
- $xz \in L_k$ y $yz \notin L_k$. Son distinguibles dos a dos. ■

La deuda de la semana 2

L_k: el k-ésimo símbolo desde el final es 1. NFA: $k+1$ estados.

Teorema

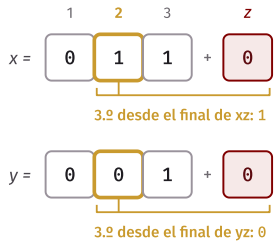
Todo DFA que reconoce L_k tiene al menos 2^k estados.

- S: las 2^k cadenas de longitud k.
- Si $x \neq y$, difieren en alguna posición i. Digamos $x_i = 1$, $y_i = 0$.
- Tomamos $z = 0^{i-1}$. En xz, de longitud $k+i-1$, el k-ésimo desde el final es la posición i.
- $xz \in L_k$ y $yz \notin L_k$. Son distinguibles dos a dos. ■

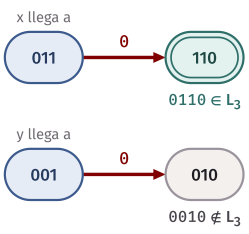
El autómata que recuerda los últimos k símbolos tiene 2^k estados: la cota se alcanza.

Con $k = 3$: 011 y 001

LAS CADENAS Y EL SUFIJO



EN EL DFA DE 8 ESTADOS



Difieren en la posición i = 2, y z = 0^{i-1} = 0 deja esa posición a distancia 3 del final.

Un DFA que llevara 011 y 001 al mismo estado respondería igual a 0110 y a 0010, y se equivocaría en una.

Comprobado con $k = 3$

terminal

```
$ build/fooling_set machines/last_3.dfa 000 001 010 011 100 101 110 111
sufijo más corto que distingue a cada par (fila, columna):
       001  010  011  100  101  110  111
  000  00   0    0    ε    ε    ε    ε
  001  .    0    0    ε    ε    ε    ε
  010  .    .    00   ε    ε    ε    ε
  011  .    .    .    ε    ε    ε    ε
  100  .    .    .    .    00   0    0
  101  .    .    .    .    .    0    0
  110  .    .    .    .    .    .    00
8 cadenas distinguibles dos a dos: todo DFA de este lenguaje tiene al menos 8 estados (este tiene 8)
```

- Difieren en la posición 1: ε. En la 2: `0`. Solo en la 3: `00`.

Comprobado con $k = 3$

terminal

```
$ build/fooling_set machines/last_3.dfa 000 001 010 011 100 101 110 111
sufijo más corto que distingue a cada par (fila, columna):
       001  010  011  100  101  110  111
  000  00   0    0    ε    ε    ε    ε
  001  .    0    0    ε    ε    ε    ε
  010  .    .    00   ε    ε    ε    ε
  011  .    .    .    ε    ε    ε    ε
  100  .    .    .    .    00   0    0
  101  .    .    .    .    .    0    0
  110  .    .    .    .    .    .    00
8 cadenas distinguibles dos a dos: todo DFA de este lenguaje tiene al menos 8 estados (este tiene 8)
```

- Difieren en la posición 1: ε. En la 2: `0`. Solo en la 3: `00`.
- **En la práctica:** "una `a` seguida de exactamente 20 caracteres" tiene un NFA de 22 estados y un DFA mínimo de más de un millón.

SECCIÓN 06

El teorema de Myhill-Nerode

06

Las clases de equivalencia

- $\equiv_L$ es una relación de equivalencia: parte a Σ^* en **clases**. $[x]$ es la clase de x.

terminal

```
$ build/classes machines/ends_01_seven.dfa 3
clases de equivalencia: 3
  [ε] (fuera de L): ε 1 11 011 111
  [0] (fuera de L): 0 00 10 000 010 100 110
  [01] (en L): 01 001 101
```

Termina en `01`: tres clases. Son los tres hechos que anotamos al diseñar el autómata en la semana 1.

Las clases de equivalencia

- $\equiv_L$ es una relación de equivalencia: parte a Σ^* en **clases**. $[x]$ es la clase de x.
- Si $x \equiv_L y$:
  - $x \in L \iff y \in L$;
  - $xa \equiv_L ya$ para todo símbolo a.

terminal

```
$ build/classes machines/ends_01_seven.dfa 3
clases de equivalencia: 3
  [ε] (fuera de L): ε 1 11 011 111
  [0] (fuera de L): 0 00 10 000 010 100 110
  [01] (en L): 01 001 101
```

Termina en `01`: tres clases. Son los tres hechos que anotamos al diseñar el autómata en la semana 1.

Las clases de equivalencia

- $\equiv_L$ es una relación de equivalencia: parte a Σ^* en **clases**. $[x]$ es la clase de x.
- Si $x \equiv_L y$:
  - $x \in L \iff y \in L$;
  - $xa \equiv_L ya$ para todo símbolo a.
- Las clases se comportan como los **estados** de un DFA.

terminal

```
$ build/classes machines/ends_01_seven.dfa 3
clases de equivalencia: 3
  [ε] (fuera de L): ε 1 11 011 111
  [0] (fuera de L): 0 00 10 000 010 100 110
  [01] (en L): 01 001 101
```

Termina en `01`: tres clases. Son los tres hechos que anotamos al diseñar el autómata en la semana 1.

El teorema de Myhill-Nerode

Teorema

1. Si un DFA con n estados reconoce L, entonces $\equiv_L$ tiene a lo sumo n clases.
2. Si $\equiv_L$ tiene k clases, hay un DFA con exactamente k estados que reconoce L.

El teorema de Myhill-Nerode

Teorema

1. Si un DFA con n estados reconoce L, entonces $\equiv_L$ tiene a lo sumo n clases.
2. Si $\equiv_L$ tiene k clases, hay un DFA con exactamente k estados que reconoce L.

- L es regular **si y solo si** $\equiv_L$ tiene finitas clases.
- Ese número es el de estados del DFA mínimo.

El teorema de Myhill-Nerode

Teorema

1. Si un DFA con n estados reconoce L, entonces $\equiv_L$ tiene a lo sumo n clases.
2. Si $\equiv_L$ tiene k clases, hay un DFA con exactamente k estados que reconoce L.

- L es regular **si y solo si** $\equiv_L$ tiene finitas clases.
- Ese número es el de estados del DFA mínimo.
- (1) es la cota inferior: $n + 1$ clases darían $n + 1$ cadenas distinguibles dos a dos.

El teorema de Myhill-Nerode

Teorema

1. Si un DFA con n estados reconoce L, entonces $\equiv_L$ tiene a lo sumo n clases.
2. Si $\equiv_L$ tiene k clases, hay un DFA con exactamente k estados que reconoce L.

- L es regular **si y solo si** $\equiv_L$ tiene finitas clases.
- Ese número es el de estados del DFA mínimo.
- (1) es la cota inferior: $n + 1$ clases darían $n + 1$ cadenas distinguibles dos a dos.
- (2) es una construcción.

El teorema de Myhill-Nerode

Teorema

1. Si un DFA con n estados reconoce L, entonces $\equiv_L$ tiene a lo sumo n clases.
2. Si $\equiv_L$ tiene k clases, hay un DFA con exactamente k estados que reconoce L.

- L es regular **si y solo si** $\equiv_L$ tiene finitas clases.
- Ese número es el de estados del DFA mínimo.
- (1) es la cota inferior: $n + 1$ clases darían $n + 1$ cadenas distinguibles dos a dos.
- (2) es una construcción.

A diferencia del lema de bombeo, vale en las dos direcciones: también demuestra que un lenguaje **sí** es regular.

El autómata de las clases

$$Q = \{[x] : x \in \Sigma^*\}, \qquad \delta([x], a) = [xa], \qquad q_0 = [\varepsilon], \qquad F = \{[x] : x \in L\}$$

El autómata de las clases

$$Q = \{[x] : x \in \Sigma^*\}, \qquad \delta([x], a) = [xa], \qquad q_0 = [\varepsilon], \qquad F = \{[x] : x \in L\}$$

- **Bien definido:** si $[x] = [y]$, entonces $[xa] = [ya]$, y $x \in L \iff y \in L$.

El autómata de las clases

$$Q = \{[x] : x \in \Sigma^*\}, \qquad \delta([x], a) = [xa], \qquad q_0 = [\varepsilon], \qquad F = \{[x] : x \in L\}$$

- **Bien definido:** si $[x] = [y]$, entonces $[xa] = [ya]$, y $x \in L \iff y \in L$.
- **Dónde termina cada cadena:** $\delta^*([\varepsilon], w) = [w]$, por inducción sobre $|w|$:

$$\delta^*([\varepsilon], ua) = \delta(\delta^*([\varepsilon], u), a) = \delta([u], a) = [ua].$$

El autómata de las clases

$$Q = \{[x] : x \in \Sigma^*\}, \qquad \delta([x], a) = [xa], \qquad q_0 = [\varepsilon], \qquad F = \{[x] : x \in L\}$$

- **Bien definido:** si $[x] = [y]$, entonces $[xa] = [ya]$, y $x \in L \iff y \in L$.
- **Dónde termina cada cadena:** $\delta^*([\varepsilon], w) = [w]$, por inducción sobre $|w|$:

$$\delta^*([\varepsilon], ua) = \delta(\delta^*([\varepsilon], u), a) = \delta([u], a) = [ua].$$

- **Reconoce L:** acepta $w \iff [w] \in F \iff w \in L$.

El autómata de las clases

$$Q = \{[x] : x \in \Sigma^*\}, \qquad \delta([x], a) = [xa], \qquad q_0 = [\varepsilon], \qquad F = \{[x] : x \in L\}$$

- **Bien definido:** si $[x] = [y]$, entonces $[xa] = [ya]$, y $x \in L \iff y \in L$.
- **Dónde termina cada cadena:** $\delta^*([\varepsilon], w) = [w]$, por inducción sobre $|w|$:

$$\delta^*([\varepsilon], ua) = \delta(\delta^*([\varepsilon], u), a) = \delta([u], a) = [ua].$$

- **Reconoce L:** acepta $w \iff [w] \in F \iff w \in L$.

Idea clave

Después de leer w recuerda la clase de w, y nada más. El lenguaje, y no quien diseña, decide cuántos estados hacen falta.

SECCIÓN 07

Minimizar un DFA

07

Qué estados sobran

- **Inalcanzables:** ninguna cadena llega a ellos desde q_0. Se borran.

Qué estados sobran

- **Inalcanzables:** ninguna cadena llega a ellos desde q_0. Se borran.
- **Equivalentes:** alcanzables los dos, y hacen lo mismo de ahí en adelante.

Estados equivalentes

$p \sim q$ si ningún sufijo los distingue: para toda z, $\delta^*(p, z) \in F \iff \delta^*(q, z) \in F$.

Qué estados sobran

- **Inalcanzables:** ninguna cadena llega a ellos desde q_0. Se borran.
- **Equivalentes:** alcanzables los dos, y hacen lo mismo de ahí en adelante.

Estados equivalentes

$p \sim q$ si ningún sufijo los distingue: para toda z, $\delta^*(p, z) \in F \iff \delta^*(q, z) \in F$.

- Es la misma idea, para estados: $x \equiv_{L(M)} y \iff \delta^*(q_0, x) \sim \delta^*(q_0, y)$.

Qué estados sobran

- **Inalcanzables:** ninguna cadena llega a ellos desde q_0. Se borran.
- **Equivalentes:** alcanzables los dos, y hacen lo mismo de ahí en adelante.

Estados equivalentes

$p \sim q$ si ningún sufijo los distingue: para toda z, $\delta^*(p, z) \in F \iff \delta^*(q, z) \in F$.

- Es la misma idea, para estados: $x \equiv_{L(M)} y \iff \delta^*(q_0, x) \sim \delta^*(q_0, y)$.
- Dos estados equivalentes se pueden **fusionar**.

Qué estados sobran

- **Inalcanzables:** ninguna cadena llega a ellos desde q_0. Se borran.
- **Equivalentes:** alcanzables los dos, y hacen lo mismo de ahí en adelante.

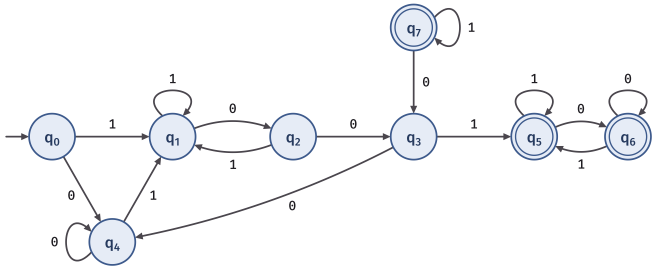
Estados equivalentes

$p \sim q$ si ningún sufijo los distingue: para toda z, $\delta^*(p, z) \in F \iff \delta^*(q, z) \in F$.

- Es la misma idea, para estados: $x \equiv_{L(M)} y \iff \delta^*(q_0, x) \sim \delta^*(q_0, y)$.
- Dos estados equivalentes se pueden **fusionar**.

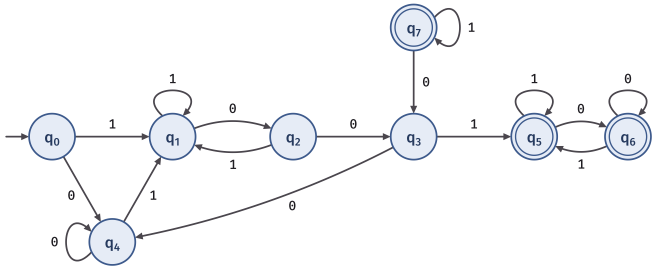
Minimizar: quitar los inalcanzables y fusionar los equivalentes.

Un autómata escrito con descuido: contiene `1001`



- q_7 es inalcanzable.

Un autómata escrito con descuido: contiene `1001`



- q_7 es inalcanzable.
- q_0 “no se leyó nada” y q_4 “no hay avance”: ¿hace falta distinguirlos?
- q_5 y q_6: ya apareció `1001`. ¿Importa el último símbolo?

“Se parecen” no es un argumento

terminal

```
$ build/distinguish machines/contains_1001.dfa '' 0
x = ε llega a q0, y = 0 llega a q4
ningún sufijo de longitud a lo sumo |Q| - 1 = 7 las distingue (255 probados): son indistinguibles
$ build/distinguish machines/contains_1001.dfa '' 1
x = ε llega a q0, y = 1 llega a q1
sufijos probados, del más corto al más largo: 9
z = 001 las distingue: xz = 001 se rechaza, yz = 1001 se acepta
```

q_0 y q_4 son equivalentes. q_0 y q_1 también “se parecen”, y **no** lo son.

Error común

Fusionar dos estados porque se parecen. Solo si **ningún** sufijo los distingue.

Refinar una partición

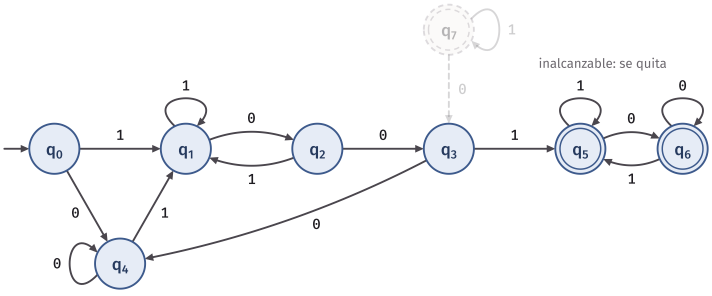
1. Quitar los inalcanzables.
2. Partición inicial: $\{F, Q - F\}$.
3. **Ronda:** dos estados siguen juntos si estaban juntos y, con cada símbolo, van al **mismo bloque**.
4. Repetir hasta que no cambie.
5. Un estado por bloque.

En bloques distintos: **ya se sabe** que son distinguibles. En el mismo: todavía no se han podido distinguir.

pseudocódigo

```
algoritmo minimizar(M)
    quitar los estados inalcanzables
    P ← {F, Q − F}, sin los bloques vacíos
    cambió ← verdadero
    mientras cambió
        para cada estado q
            firma[q] ← (bloque de q en P,
                bloque de δ(q, a) en P para cada a)
        fin para
        P' ← los estados agrupados por firma
        cambió ← P' tiene más bloques que P
        P ← P'
    fin mientras
    devolver un estado por bloque de P
fin algoritmo
```

Las rondas, en el autómata

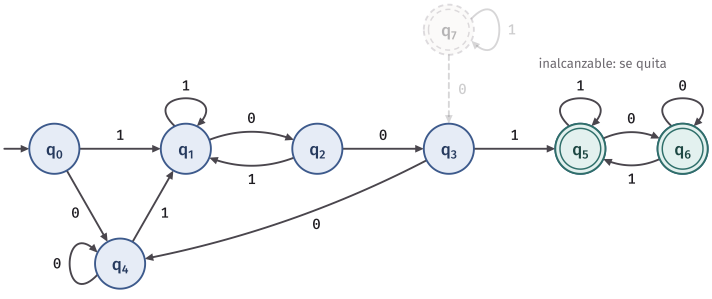


8 estados, 7 alcanzables

Primero se quitan los inalcanzables: ninguna flecha de los otros estados llega a q_7.

Quedan 7 estados.

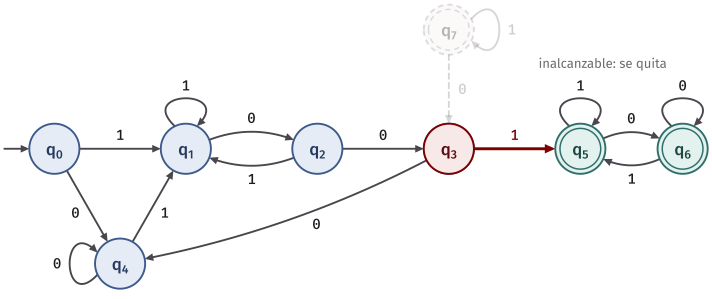
Las rondas, en el autómata



P_0: dos bloques, los que no aceptan y los que aceptan.

Ya los distingue el sufijo ε. Dentro de un bloque todavía no se ha distinguido a nadie.

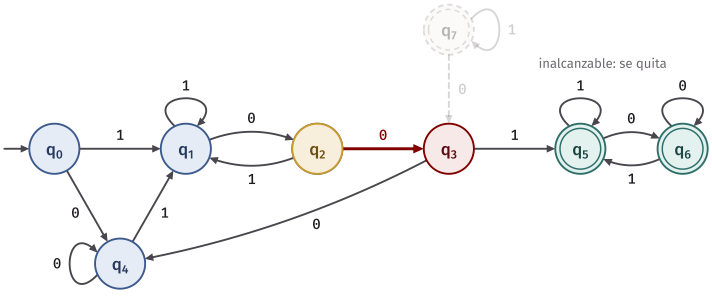
Las rondas, en el autómata



Ronda 1: con 1, q_3 va al bloque B y el resto de su bloque se queda en A. Se separa.

A q_3 lo distingue de los demás el sufijo 1.

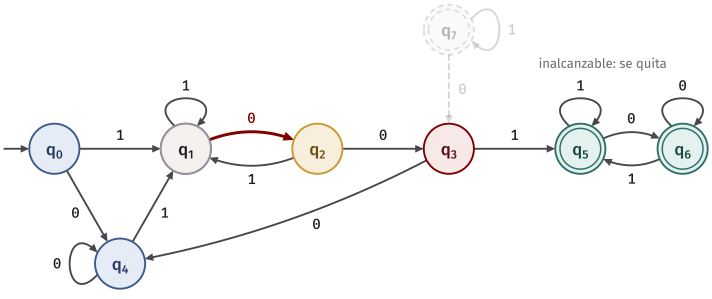
Las rondas, en el autómata



Ronda 2: con 0, q_2 va al bloque nuevo de q_3 y los demás de su bloque no. Se separa.

Lo distingue el sufijo `01`: la información viaja un paso hacia atrás por ronda.

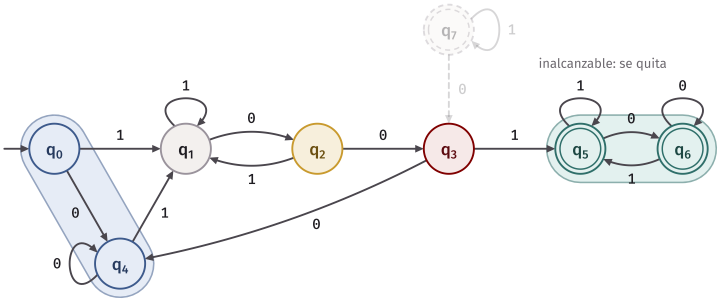
Las rondas, en el autómata



Ronda 3: con 0, q_1 va al bloque de q_2, y q_0 y q_4 no. Se separa.

Lo distingue el sufijo `001`, el que encontró `distinguish`.

Las rondas, en el autómata



Ronda 4: q_0 y q_4 van a los mismos bloques, y q_5 y q_6 también. No se separa nada: $P_4 = P_3$.

Cinco bloques: el DFA mínimo tiene 5 estados. Cada pareja se funde en uno.

El ejemplo, ronda por ronda

$P_0 \quad \{q_0, q_1, q_2, q_3, q_4\} \quad \{q_5, q_6\}$

El ejemplo, ronda por ronda

P_0 $\{q_0, q_1, q_2, q_3, q_4\}$ $\{q_5, q_6\}$
P_1 $\{q_0, q_1, q_2, q_4\}$ $\{q_3\}$ $\{q_5, q_6\}$

- Ronda 1: con 1, solo q_3 sale de su bloque. Lo separa el sufijo 1.

El ejemplo, ronda por ronda

P_0 $\{q_0, q_1, q_2, q_3, q_4\}$ $\{q_5, q_6\}$
P_1 $\{q_0, q_1, q_2, q_4\}$ $\{q_3\}$ $\{q_5, q_6\}$
P_2 $\{q_0, q_1, q_4\}$ $\{q_2\}$ $\{q_3\}$ $\{q_5, q_6\}$

- Ronda 1: con `1`, solo q_3 sale de su bloque. Lo separa el sufijo `1`.
- Ronda 2: q_2 va a q_3 con `0`, y los demás no. Sufijo `01`.

El ejemplo, ronda por ronda

P_0 $\{q_0, q_1, q_2, q_3, q_4\}$ $\{q_5, q_6\}$
P_1 $\{q_0, q_1, q_2, q_4\}$ $\{q_3\}$ $\{q_5, q_6\}$
P_2 $\{q_0, q_1, q_4\}$ $\{q_2\}$ $\{q_3\}$ $\{q_5, q_6\}$
P_3 $\{q_0, q_4\}$ $\{q_1\}$ $\{q_2\}$ $\{q_3\}$ $\{q_5, q_6\}$

- Ronda 1: con `1`, solo q_3 sale de su bloque. Lo separa el sufijo `1`.
- Ronda 2: q_2 va a q_3 con `0`, y los demás no. Sufijo `01`.
- Ronda 3: q_1 va a q_2 con `0`. Sufijo `001`: el que encontró `distinguish`.

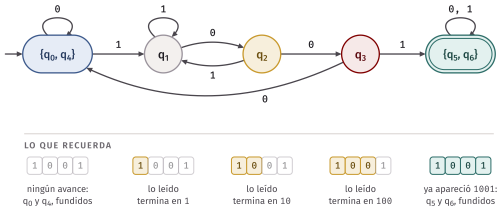
El ejemplo, ronda por ronda

P_0 $\{q_0, q_1, q_2, q_3, q_4\}$ $\{q_5, q_6\}$
P_1 $\{q_0, q_1, q_2, q_4\}$ $\{q_3\}$ $\{q_5, q_6\}$
P_2 $\{q_0, q_1, q_4\}$ $\{q_2\}$ $\{q_3\}$ $\{q_5, q_6\}$
P_3 $\{q_0, q_4\}$ $\{q_1\}$ $\{q_2\}$ $\{q_3\}$ $\{q_5, q_6\}$
P_4 $= P_3$: termina

- Ronda 1: con `1`, solo q_3 sale de su bloque. Lo separa el sufijo `1`.
- Ronda 2: q_2 va a q_3 con `0`, y los demás no. Sufijo `01`.
- Ronda 3: q_1 va a q_2 con `0`. Sufijo `001`: el que encontró `distinguish`.

La información viaja hacia atrás: después de la ronda k están separados los estados que se distinguen con sufijos de longitud a lo sumo k.

El resultado: cinco estados

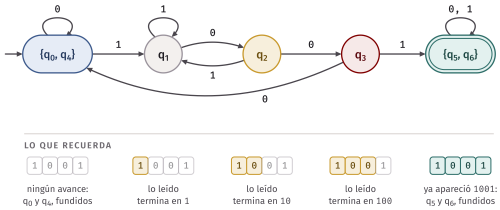


Un estado por cada avance hacia `1001`: el que habríamos diseñado en la semana 1.

terminal

```
$ build/minimize machines/contains_1001.dfa --todos | grep -E '^(estados|DFA)'
estados: 8, alcanzables: 7. No se quita ninguno (--todos)
DFA mínimo: 6 estados
```

El resultado: cinco estados



Un estado por cada avance hacia `1001`: el que habríamos diseñado en la semana 1. Sin quitar q_7 primero sale correcto, pero no mínimo:

terminal

```
$ build/minimize machines/contains_1001.dfa --todos | grep -E '^(estados|DFA)'
estados: 8, alcanzables: 7. No se quita ninguno (--todos)
DFA mínimo: 6 estados
```

Por qué funciona: el lema de la ronda

$p \sim_k q$: ningún sufijo **de longitud a lo sumo** k distingue a p de q.

Lema de la ronda

$p \sim_{k+1} q$ si y solo si $p \sim_k q$ y $\delta(p, a) \sim_k \delta(q, a)$ para todo símbolo a.

Por qué funciona: el lema de la ronda

$p \sim_k q$: ningún sufijo **de longitud a lo sumo** k distingue a p de q.

Lema de la ronda

$p \sim_{k+1} q$ si y solo si $p \sim_k q$ y $\delta(p, a) \sim_k \delta(q, a)$ para todo símbolo a.

- Un sufijo de longitud a lo sumo $k + 1$ es ε o es az con $|z| \leq k$.
- az distingue a p de q exactamente cuando z distingue a $\delta(p, a)$ de $\delta(q, a)$.

Por qué funciona: el lema de la ronda

$p \sim_k q$: ningún sufijo **de longitud a lo sumo** k distingue a p de q.

Lema de la ronda

$p \sim_{k+1} q$ si y solo si $p \sim_k q$ y $\delta(p, a) \sim_k \delta(q, a)$ para todo símbolo a.

- Un sufijo de longitud a lo sumo $k + 1$ es ε o es az con $|z| \leq k$.
- az distingue a p de q exactamente cuando z distingue a $\delta(p, a)$ de $\delta(q, a)$.
- $P_0 = \{F, Q - F\}$ es $\sim_0$: el único sufijo es ε.

Por qué funciona: el lema de la ronda

$p \sim_k q$: ningún sufijo **de longitud a lo sumo** k distingue a p de q.

Lema de la ronda

$p \sim_{k+1} q$ si y solo si $p \sim_k q$ y $\delta(p, a) \sim_k \delta(q, a)$ para todo símbolo a.

- Un sufijo de longitud a lo sumo $k + 1$ es ε o es az con $|z| \leq k$.
- az distingue a p de q exactamente cuando z distingue a $\delta(p, a)$ de $\delta(q, a)$.
- $P_0 = \{F, Q - F\}$ es $\sim_0$: el único sufijo es ε.
- La ronda aplica el lema: P_k **es la partición de** $\sim_k$**,** por inducción.

Termina, es equivalente y es mínimo

1. **Termina.** Cada ronda que cambia algo agrega un bloque, y hay a lo sumo $|Q|$.

Termina, es equivalente y es mínimo

1. **Termina.** Cada ronda que cambia algo agrega un bloque, y hay a lo sumo $|Q|$.
2. **La partición final es $\sim$.** Si una ronda no cambia nada, ninguna posterior lo hará: la regla es la misma y el dato es el mismo.

Termina, es equivalente y es mínimo

1. **Termina.** Cada ronda que cambia algo agrega un bloque, y hay a lo sumo $|Q|$.
2. **La partición final es $\sim$.** Si una ronda no cambia nada, ninguna posterior lo hará: la regla es la misma y el dato es el mismo.
3. **Reconoce el mismo lenguaje.** $\delta'([p], a) = [\delta(p, a)]$ está bien definida, y $\delta'^*([p], w) = [\delta^*(p, w)]$.

Termina, es equivalente y es mínimo

1. **Termina.** Cada ronda que cambia algo agrega un bloque, y hay a lo sumo $|Q|$.
2. **La partición final es $\sim$.** Si una ronda no cambia nada, ninguna posterior lo hará: la regla es la misma y el dato es el mismo.
3. **Reconoce el mismo lenguaje.** $\delta'([p], a) = [\delta(p, a)]$ está bien definida, y $\delta'^*([p], w) = [\delta^*(p, w)]$.
4. **Es mínimo.** Una cadena x_p que llegue a cada bloque: son distinguibles dos a dos. Por la cota inferior, ningún DFA tiene menos estados.

Termina, es equivalente y es mínimo

1. **Termina.** Cada ronda que cambia algo agrega un bloque, y hay a lo sumo $|Q|$.
2. **La partición final es $\sim$.** Si una ronda no cambia nada, ninguna posterior lo hará: la regla es la misma y el dato es el mismo.
3. **Reconoce el mismo lenguaje.** $\delta'([p], a) = [\delta(p, a)]$ está bien definida, y $\delta'^*([p], w) = [\delta^*(p, w)]$.
4. **Es mínimo.** Una cadena x_p que llegue a cada bloque: son distinguibles dos a dos. Por la cota inferior, ningún DFA tiene menos estados.

Dónde se usa el paso 1

Para un estado inalcanzable no existe la cadena x_p. Por eso con q_7 salía un estado de más.

Cuánto cuesta

- Una ronda: n firmas de $|\Sigma| + 1$ números. A lo sumo n rondas: Moore cuesta $O(n^2 \cdot |\Sigma|)$.
- Muchas veces las rondas son pocas: el autómata de L_k necesita $k - 1$.
- Hopcroft: $O(n \log n \cdot |\Sigma|)$. Calcula la misma partición; no lo estudiamos.

terminal

```
$ build/kth_from_end --tabla 6
 k  estados del NFA  estados del DFA  después de minimizar  rondas con cambios
 1                2                2                     2                   0
 2                3                4                     4                   1
 3                4                8                     8                   2
 4                5               16                    16                   3
 5                6               32                    32                   4
 6                7               64                    64                   5
```

Al minimizar el autómata de 2^k estados no se fusiona ninguno: es lo que demostramos.

SECCIÓN 08

El DFA mínimo es único

08

Único, salvo los nombres

Teorema

Si $\equiv_L$ tiene k clases, todo DFA con k estados que reconoce L es igual, salvo los nombres de los estados, al autómata de las clases.

Único, salvo los nombres

Teorema

Si $\equiv_L$ tiene k clases, todo DFA con k estados que reconoce L es igual, salvo los nombres de los estados, al autómata de las clases.

- Todos sus estados son alcanzables: si no, habría un DFA con menos de k.

Único, salvo los nombres

Teorema

Si $\equiv_L$ tiene k clases, todo DFA con k estados que reconoce L es igual, salvo los nombres de los estados, al autómata de las clases.

- Todos sus estados son alcanzables: si no, habría un DFA con menos de k.
- $h(q) = [x]$, donde x es cualquier cadena que llega a q. No depende de x.

Único, salvo los nombres

Teorema

Si $\equiv_L$ tiene k clases, todo DFA con k estados que reconoce L es igual, salvo los nombres de los estados, al autómata de las clases.

- Todos sus estados son alcanzables: si no, habría un DFA con menos de k.
- $h(q) = [x]$, donde x es cualquier cadena que llega a q. No depende de x.
- Es sobreyectiva entre dos conjuntos de k elementos: es una biyección.

Único, salvo los nombres

Teorema

Si $\equiv_L$ tiene k clases, todo DFA con k estados que reconoce L es igual, salvo los nombres de los estados, al autómata de las clases.

- Todos sus estados son alcanzables: si no, habría un DFA con menos de k.
- $h(q) = [x]$, donde x es cualquier cadena que llega a q. No depende de x.
- Es sobreyectiva entre dos conjuntos de k elementos: es una biyección.
- Respeta el inicial, las transiciones y la aceptación: $h(\delta(q, a)) = [xa]$.

Único, salvo los nombres

Teorema

Si $\equiv_L$ tiene k clases, todo DFA con k estados que reconoce L es igual, salvo los nombres de los estados, al autómata de las clases.

- Todos sus estados son alcanzables: si no, habría un DFA con menos de k.
- $h(q) = [x]$, donde x es cualquier cadena que llega a q. No depende de x.
- Es sobreyectiva entre dos conjuntos de k elementos: es una biyección.
- Respeta el inicial, las transiciones y la aceptación: $h(\delta(q, a)) = [xa]$.

Vale para los DFA. Un lenguaje puede tener varios NFA mínimos distintos.

Decidir si dos DFA son equivalentes

Minimizar los dos y comprobar si los mínimos son iguales salvo los nombres.

terminal

```
$ build/equivalent machines/ends_01_seven.dfa machines/ends_01.dfa
machines/ends_01_seven.dfa: 7 estados, mínimo de 3
machines/ends_01.dfa: 3 estados, mínimo de 3
los dos mínimos son el mismo autómata salvo los nombres: reconocen el mismo lenguaje
```

terminal

```
$ build/equivalent machines/contains_101.dfa machines/contains_1001.dfa
machines/contains_101.dfa: 4 estados, mínimo de 4
machines/contains_1001.dfa: 8 estados, mínimo de 5
los mínimos son distintos: no reconocen el mismo lenguaje
la cadena más corta que los separa: 101 (la acepta el primero)
```

Siempre termina y siempre responde bien. En la semana 9: $\mathrm{EQ}_{\mathrm{DFA}}$ es decidible.

SECCIÓN 09

Opcional: memoria en streaming

09

Opcional: una cota inferior de memoria

Un algoritmo de *streaming* lee los datos una vez, en orden, con poca memoria.

El algoritmo	**El DFA**
Un contenido de la memoria	Un estado: hay $\leq 2^s$
Actualizar con un dato	δ
Contenidos con respuesta sí	F

Opcional: una cota inferior de memoria

Un algoritmo de *streaming* lee los datos una vez, en orden, con poca memoria.

El algoritmo	El DFA
Un contenido de la memoria	Un estado: hay $\leq 2^s$
Actualizar con un dato	δ
Contenidos con respuesta sí	F

Teorema

Con s bits y k cadenas distinguibles dos a dos: $s \geq \log_2 k$.

Opcional: una cota inferior de memoria

Un algoritmo de *streaming* lee los datos una vez, en orden, con poca memoria.

El algoritmo	El DFA
Un contenido de la memoria	Un estado: hay $\leq 2^s$
Actualizar con un dato	δ
Contenidos con respuesta sí	F

Teorema

Con s bits y k cadenas distinguibles dos a dos: $s \geq \log_2 k$.

- ¿Hubo un error hace exactamente k segundos? Es L_k.

Opcional: una cota inferior de memoria

Un algoritmo de *streaming* lee los datos una vez, en orden, con poca memoria.

El algoritmo	El DFA
Un contenido de la memoria	Un estado: hay $\leq 2^s$
Actualizar con un dato	δ
Contenidos con respuesta sí	F

Teorema

Con s bits y k cadenas distinguibles dos a dos: $s \geq \log_2 k$.

- ¿Hubo un error hace exactamente k segundos? Es L_k.
- Cota inferior: 2^k cadenas distinguibles, $s \geq k$ bits.

Opcional: una cota inferior de memoria

Un algoritmo de *streaming* lee los datos una vez, en orden, con poca memoria.

El algoritmo	El DFA
Un contenido de la memoria	Un estado: hay $\leq 2^s$
Actualizar con un dato	δ
Contenidos con respuesta sí	F

Teorema

Con s bits y k cadenas distinguibles dos a dos: $s \geq \log_2 k$.

- ¿Hubo un error hace exactamente k segundos? Es L_k.
- Cota inferior: 2^k cadenas distinguibles, $s \geq k$ bits.
- Cota superior: un registro con los últimos k bits. **Exactamente k bits.**

Opcional: una cota inferior de memoria

Un algoritmo de *streaming* lee los datos una vez, en orden, con poca memoria.

El algoritmo	El DFA
Un contenido de la memoria	Un estado: hay $\leq 2^s$
Actualizar con un dato	δ
Contenidos con respuesta sí	F

Teorema

Con s bits y k cadenas distinguibles dos a dos: $s \geq \log_2 k$.

- ¿Hubo un error hace exactamente k segundos? Es L_k.
- Cota inferior: 2^k cadenas distinguibles, $s \geq k$ bits.
- Cota superior: un registro con los últimos k bits. **Exactamente k bits.**

Una cota inferior dice cuándo dejar de buscar el algoritmo exacto y cambiar la pregunta.

SECCIÓN 10

Cierre

10

Qué herramienta sirve para qué

Demostrar que L…	**Sirve**	**No sirve**
es regular	Un DFA, un NFA, una expresión regular; cierre; finitas clases de $\equiv_L$	El lema de bombeo
no es regular	El lema de bombeo; cierre hasta un lenguaje no regular; infinitas cadenas distinguibles	“No se me ocurre un DFA”
necesita k estados	k cadenas distinguibles dos a dos	Probar con autómatas pequeños

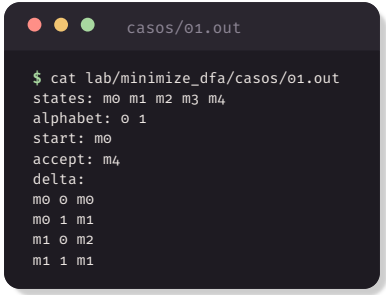
- Con esto termina la parte 1: sabemos qué es regular, cómo describirlo y dónde termina.

Laboratorio: Práctica 1. Opcional: minimizar un DFA

La sesión es para la **Práctica 1** (semanas 1 a 3). Como ejercicio opcional:

1. Leer un DFA y escribir el DFA mínimo equivalente.
2. Eliminar primero los estados inalcanzables.
3. Refinar $\{F, Q - F\}$ hasta que no cambie.
4. El verificador comprueba que la salida es equivalente y que tiene el número mínimo de estados.
5. Minimizar el DFA de la semana 2 para $k = 6$: entran 64 estados y salen 64.

Dónde: `code/week-04/lab/minimize_dfa`. Con 2000 estados, decidir cada par por separado da `TLE`.



Las primeras líneas. Cualquier otro nombre para los estados también vale.

Para la próxima clase

- Leer los apuntes de la semana 4. El ejercicio de minimización es opcional.
- Preguntas de repaso de la guía: $\{a^n b^m : n \leq m\}$ con el lema y con cadenas distinguibles; por qué el lema no demuestra que un lenguaje es regular; minimizar a mano un DFA de seis estados; cuántos estados necesitan los múltiplos de 5.
- Próxima semana: gramáticas libres de contexto. Una manera de describir $\{0^n 1^n\}$, los paréntesis balanceados y la estructura anidada.

Pregunta

Un autómata finito no reconoce $\{0^n 1^n\}$. ¿Qué es lo mínimo que habría que agregarle para que pueda?