

UNIVERSIDAD NACIONAL DE INGENIERÍA

Teoría de la Computación

Reducciones

Teorema de Rice, historias de cómputo y el problema de correspondencia de Post

Gerson Garrido

Semana 10

Hoy

Sesión A

1. Reducir: la idea y la dirección
2. $\mathrm{HALT}_{\mathrm{TM}}$, por reducción
3. Fijar la entrada dentro de la máquina: E_{TM}
4. $\mathrm{REGULAR}_{\mathrm{TM}}$ y $\mathrm{EQ}_{\mathrm{TM}}$
5. Reducción por función
6. El teorema de Rice, y lo que no dice

Sesión B

7. Problemas que no hablan de máquinas
8. Historias de cómputo
9. El problema de correspondencia de Post
10. PCP es indecidible
11. $\mathrm{ALL}_{\mathrm{CFG}}$, $\mathrm{EQ}_{\mathrm{CFG}}$ y la ambigüedad
12. El contraste con los autómatas finitos

Lectura: apuntes de la semana 10. Apoyo: Sipser, sec. 5.1, 5.2 y 5.3; Erickson, *Models of Computation*, nota 7.

SECCIÓN 01

Reducir: la idea y la dirección

01

De dónde venimos

- Semana 9: $A_{\mathrm{TM}} = \{ \langle M, w \rangle : M \text{ acepta } w \}$ es **indecidible**.
- Se demostró por diagonalización: un decididor H, una máquina D que lo contradice, y D con $\langle D \rangle$.

De dónde venimos

- Semana 9: $A_{\mathrm{TM}} = \{ \langle M, w \rangle : M \text{ acepta } w \}$ es **indecidible**.
- Se demostró por diagonalización: un decididor H, una máquina D que lo contradice, y D con $\langle D \rangle$.
- Inventar una diagonal para cada problema nuevo no es práctico.

De dónde venimos

- Semana 9: $A_{\mathrm{TM}} = \{ \langle M, w \rangle : M \text{ acepta } w \}$ es **indecidible**.
- Se demostró por diagonalización: un decididor H, una máquina D que lo contradice, y D con $\langle D \rangle$.
- Inventar una diagonal para cada problema nuevo no es práctico.

Reducción *(reduction)*

Una reducción de A a B es una manera de resolver A usando una solución de B.

De dónde venimos

- Semana 9: $A_{\mathrm{TM}} = \{ \langle M, w \rangle : M \text{ acepta } w \}$ es **indecidible**.
- Se demostró por diagonalización: un decididor H, una máquina D que lo contradice, y D con $\langle D \rangle$.
- Inventar una diagonal para cada problema nuevo no es práctico.

Reducción ***(reduction)***

Una reducción de A a B es una manera de resolver A usando una solución de B.

- Si B se puede resolver, A se puede resolver. (el uso de Algoritmos)

De dónde venimos

- Semana 9: $A_{\mathrm{TM}} = \{ \langle M, w \rangle : M \text{ acepta } w \}$ es **indecidible**.
- Se demostró por diagonalización: un decididor H, una máquina D que lo contradice, y D con $\langle D \rangle$.
- Inventar una diagonal para cada problema nuevo no es práctico.

Reducción ***(reduction)***

Una reducción de A a B es una manera de resolver A usando una solución de B.

- Si B se puede resolver, A se puede resolver. (el uso de Algoritmos)
- Si A no se puede resolver, B tampoco. (el uso de hoy)

El plan de todas las demostraciones

Para demostrar que B es indecidible:

1. Suponer que una máquina R **decide** B.

El plan de todas las demostraciones

Para demostrar que B es indecidible:

1. Suponer que una máquina R **decide** B.
2. Con R como subrutina, construir S que decide A_{TM}.

El plan de todas las demostraciones

Para demostrar que B es indecidible:

1. Suponer que una máquina R **decide** B.
2. Con R como subrutina, construir S que decide A_{TM}.
3. A_{TM} es indecidible: S no existe. Entonces R tampoco.

El plan de todas las demostraciones

Para demostrar que B es indecidible:

1. Suponer que una máquina R **decide** B.
2. Con R como subrutina, construir S que decide A_{TM}.
3. A_{TM} es indecidible: S no existe. Entonces R tampoco.

> **La dirección**
>
> Se reduce **desde** un indecidible conocido **hacia** B: "B es al menos tan difícil como el conocido". El problema conocido es el que se resuelve; el problema nuevo es la subrutina.

Tres avisos sobre estas demostraciones

- **Son por contradicción.** *R* no existe, y se usa como si existiera. No hay que explicar cómo trabaja: solo que se detiene siempre y responde bien.

Tres avisos sobre estas demostraciones

- **Son por contradicción.** *R* no existe, y se usa como si existiera. No hay que explicar cómo trabaja: solo que se detiene siempre y responde bien.
- **Las máquinas se describen en alto nivel,** como en la semana 9: pasos numerados, entre comillas. Una entrada que no tiene la forma pedida se rechaza.

Tres avisos sobre estas demostraciones

- **Son por contradicción.** R no existe, y se usa como si existiera. No hay que explicar cómo trabaja: solo que se detiene siempre y responde bien.
- **Las máquinas se describen en alto nivel,** como en la semana 9: pasos numerados, entre comillas. Una entrada que no tiene la forma pedida se rechaza.
- **Las máquinas construyen máquinas.** "Construir M'" es *escribir la descripción* de M' a partir de la de M, como un programa que genera código.

> **La diferencia que importa hoy**
>
> Construir una máquina no es ejecutarla.

La dirección contraria no demuestra nada

- A_{DFA}: ¿un DFA B acepta w? Es **decidible**: se simula.

La dirección contraria no demuestra nada

- A_{DFA}: ¿un DFA B acepta w? Es **decidible**: se simula.
- Y se reduce a A_{TM} sin esfuerzo: un DFA se convierte en una máquina de Turing M_B que lee de izquierda a derecha, y B acepta $w \iff M_B$ acepta w.

La dirección contraria no demuestra nada

- A_{DFA}: ¿un DFA B acepta w? Es **decidible**: se simula.
- Y se reduce a A_{TM} sin esfuerzo: un DFA se convierte en una máquina de Turing M_B que lee de izquierda a derecha, y B acepta $w \iff M_B$ acepta w.
- Con un decididor de A_{TM} decidiríamos A_{DFA}. Cierto, y no dice nada malo de A_{DFA}.

La dirección contraria no demuestra nada

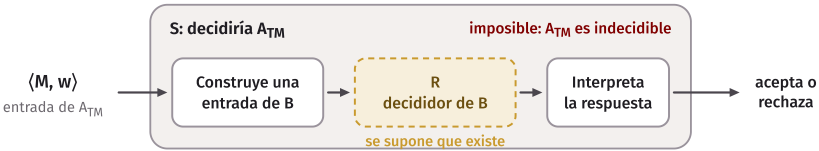
- A_{DFA}: ¿un DFA B acepta w? Es **decidible**: se simula.
- Y se reduce a A_{TM} sin esfuerzo: un DFA se convierte en una máquina de Turing M_B que lee de izquierda a derecha, y B acepta $w \iff M_B$ acepta w.
- Con un decididor de A_{TM} decidiríamos A_{DFA}. Cierto, y no dice nada malo de A_{DFA}.

Error común: reducir al revés

Reducir B a A_{TM} solo muestra que B no es más difícil que A_{TM}. Pregunten siempre: ¿qué máquina *supongo* que existe (la del problema nuevo) y cuál *construyo* (la del problema imposible)?

Las dos direcciones

Para demostrar que B es indecidible: desde A_{TM}, el conocido, hacia B, el nuevo



Se supone la máquina del problema nuevo y se construye la del imposible. Contradicción: R no existe.

Al revés, desde B hacia A_{TM}: no demuestra nada



Así se reduce A_{DFA} a A_{TM}, y A_{DFA} es decidible. Solo dice que B no es más difícil que A_{TM}.

SECCIÓN 02

El problema de la parada, por reducción

02

$\mathrm{HALT}_{\mathrm{TM}}$: el intento que falla

El problema de la parada ***(halting problem)***

$\mathrm{HALT}_{\mathrm{TM}} = \{ \langle M, w \rangle : M \text{ se detiene con } w \}$ (aceptando o rechazando)

- Para decidir A_{TM} uno quiere **simular** M con w y responder lo mismo.

$\mathrm{HALT}_{\mathrm{TM}}$: el intento que falla

El problema de la parada ***(halting problem)***

$\mathrm{HALT}_{\mathrm{TM}} = \{ \langle M, w \rangle : M \text{ se detiene con } w \}$ (aceptando o rechazando)

- Para decidir A_{TM} uno quiere **simular** M con w y responder lo mismo.
- Falla en un solo caso: si M no se detiene, la simulación tampoco.

$\mathrm{HALT}_{\mathrm{TM}}$: el intento que falla

El problema de la parada ***(halting problem)***

$\mathrm{HALT}_{\mathrm{TM}} = \{ \langle M, w \rangle : M \text{ se detiene con } w \}$ (aceptando o rechazando)

- Para decidir A_{TM} uno quiere **simular** M con w y responder lo mismo.
- Falla en un solo caso: si M no se detiene, la simulación tampoco.
- Si alguien avisara de antemano "M no se va a detener con w", ese caso desaparece.

$\mathrm{HALT}_{\mathrm{TM}}$: el intento que falla

El problema de la parada ***(halting problem)***

$\mathrm{HALT}_{\mathrm{TM}} = \{ \langle M, w \rangle : M \text{ se detiene con } w \}$ (aceptando o rechazando)

- Para decidir A_{TM} uno quiere **simular** M con w y responder lo mismo.
- Falla en un solo caso: si M no se detiene, la simulación tampoco.
- Si alguien avisara de antemano "M no se va a detener con w", ese caso desaparece.
- Un decididor de $\mathrm{HALT}_{\mathrm{TM}}$ es justo ese aviso.

$\mathrm{HALT}_{\mathrm{TM}}$ es indecidible

Reducción desde A_{TM} hacia $\mathrm{HALT}_{\mathrm{TM}}$. Supongamos que R decide $\mathrm{HALT}_{\mathrm{TM}}$.

S = **"Con la entrada $\langle M, w \rangle$:**

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

$\mathrm{HALT}_{\mathrm{TM}}$ es indecidible

Reducción desde A_{TM} hacia $\mathrm{HALT}_{\mathrm{TM}}$. Supongamos que R decide $\mathrm{HALT}_{\mathrm{TM}}$.

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

- **Se detiene siempre:** el paso 2 solo simula cuando R garantizó que la simulación termina.

$\mathrm{HALT}_{\mathrm{TM}}$ es indecidible

Reducción desde A_{TM} hacia $\mathrm{HALT}_{\mathrm{TM}}$. Supongamos que R decide $\mathrm{HALT}_{\mathrm{TM}}$.

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

- **Se detiene siempre:** el paso 2 solo simula cuando R garantizó que la simulación termina.
- M **acepta** w**:** R acepta, la simulación termina aceptando, S acepta.

$\mathrm{HALT}_{\mathrm{TM}}$ es indecidible

Reducción desde A_{TM} hacia $\mathrm{HALT}_{\mathrm{TM}}$. Supongamos que R decide $\mathrm{HALT}_{\mathrm{TM}}$.

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

- **Se detiene siempre:** el paso 2 solo simula cuando R garantizó que la simulación termina.
- M **acepta** w**:** R acepta, la simulación termina aceptando, S acepta.
- M **no acepta** w**:** si no se detiene, S rechaza en el paso 1; si rechaza, en el paso 3.

$\mathrm{HALT}_{\mathrm{TM}}$ es indecidible

Reducción desde A_{TM} hacia $\mathrm{HALT}_{\mathrm{TM}}$. Supongamos que R decide $\mathrm{HALT}_{\mathrm{TM}}$.

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

- **Se detiene siempre:** el paso 2 solo simula cuando R garantizó que la simulación termina.
- M **acepta** w**:** R acepta, la simulación termina aceptando, S acepta.
- M **no acepta** w**:** si no se detiene, S rechaza en el paso 1; si rechaza, en el paso 3.
- S decidiría A_{TM}. Contradicción: R no existe.

SECCIÓN 03

Fijar la entrada dentro de la máquina

03

E_{TM}: ¿el lenguaje de una máquina es vacío?

El problema

$E_{\mathrm{TM}} = \{ \langle M \rangle : L(M) = \emptyset \}$

- Una instancia de A_{TM} es un par $\langle M, w \rangle$. Una de E_{TM} es **solo una máquina**.

E_{TM}: ¿el lenguaje de una máquina es vacío?

El problema

$E_{\mathrm{TM}} = \{ \langle M \rangle : L(M) = \emptyset \}$

- Una instancia de A_{TM} es un par $\langle M, w \rangle$. Una de E_{TM} es **solo una máquina**.
- Hay que meter w en algún sitio: dentro de la máquina.

E_{TM}: ¿el lenguaje de una máquina es vacío?

El problema

$E_{\mathrm{TM}} = \{ \langle M \rangle : L(M) = \emptyset \}$

- Una instancia de A_{TM} es un par $\langle M, w \rangle$. Una de E_{TM} es **solo una máquina**.
- Hay que meter w en algún sitio: dentro de la máquina.

M_w = "Con la entrada x:

1. Si $x \neq w$, *rechazar*.
2. Si $x = w$, correr M con w. Si M acepta, *aceptar*."

E_{TM}: ¿el lenguaje de una máquina es vacío?

El problema

$E_{\mathrm{TM}} = \{ \langle M \rangle : L(M) = \emptyset \}$

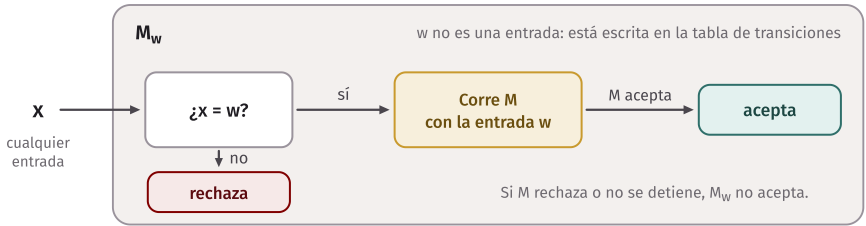
- Una instancia de A_{TM} es un par $\langle M, w \rangle$. Una de E_{TM} es **solo una máquina**.
- Hay que meter w en algún sitio: dentro de la máquina.

M_w = "Con la entrada x:

1. Si $x \neq w$, *rechazar*.
2. Si $x = w$, correr M con w. Si M acepta, *aceptar*."

$$L(M_w) = \{w\} \text{ si } M \text{ acepta } w, \qquad L(M_w) = \emptyset \text{ si no.}$$

M_w y su lenguaje



¿Cuál es el lenguaje de M_w? La única cadena que puede aceptar es w.

Si M acepta w	$L(M_w) = \{w\}$ no es vacío	$\langle M_w \rangle \notin E_{TM}$
Si M no acepta w	$L(M_w) = \emptyset$ también si M no se detiene	$\langle M_w \rangle \in E_{TM}$

«¿M acepta w?» quedó convertida en «¿es vacío el lenguaje de M_w?», con la respuesta invertida.

La construcción es un programa

build/fix_input

machines/starts_01.tm 01 (desde start:)

```
start: c0
accept: m.q_accept
reject: m.q_reject
delta:
c0 0 -> c1 @ R
c1 1 -> c2 1 R
c2 _ -> back _ L
back 0 -> back 0 L
back 1 -> back 1 L
back @ -> m.q0 0 L
m.q0 0 -> m.q1 0 R
m.q1 1 -> m.q2 1 L
m.q2 0 -> m.q_accept 0 R
```

- *M* acepta las cadenas que empiezan con `01`.
- `c0`, `c1`, `c2` comparan la entrada con *w*: la cadena *w* está en esas líneas.
- `back` vuelve al principio y entrega el control a *M*.
- Las últimas tres líneas son *M*.

Lo importante

`fix_input` **no ejecuta** a *M*. Lee una descripción y escribe otra. Termina siempre.

E_{TM} es indecidible

Reducción desde A_{TM} hacia E_{TM}. Supongamos que R decide E_{TM}.

> S = **"Con la entrada $\langle M, w \rangle$:**
>
> 1. Construir la descripción de M_w.
> 2. Correr R con $\langle M_w \rangle$. Si R acepta, *rechazar*. Si R rechaza, *aceptar*."

E_{TM} es indecidible

Reducción desde A_{TM} hacia E_{TM}. Supongamos que R decide E_{TM}.

> S = **"Con la entrada $\langle M, w \rangle$:**
>
> 1. Construir la descripción de M_w.
> 2. Correr R con $\langle M_w \rangle$. Si R acepta, *rechazar*. Si R rechaza, *aceptar*."

- **Se detiene siempre:** escribe un texto y corre un decididor. No corre M ni M_w.

E_{TM} es indecidible

Reducción desde A_{TM} hacia E_{TM}. Supongamos que R decide E_{TM}.

> S = **"Con la entrada $\langle M, w \rangle$:**
>
> 1. Construir la descripción de M_w.
> 2. Correr R con $\langle M_w \rangle$. Si R acepta, *rechazar*. Si R rechaza, *aceptar*."

- **Se detiene siempre:** escribe un texto y corre un decididor. No corre M ni M_w.
- M **acepta** w**:** $L(M_w) = \{w\} \neq \emptyset$**,** R rechaza, S acepta.

E_{TM} es indecidible

Reducción desde A_{TM} hacia E_{TM}. Supongamos que R decide E_{TM}.

> S = **"Con la entrada $\langle M, w \rangle$:**
>
> 1. Construir la descripción de M_w.
> 2. Correr R con $\langle M_w \rangle$. Si R acepta, *rechazar*. Si R rechaza, *aceptar*."

- **Se detiene siempre:** escribe un texto y corre un decididor. No corre M ni M_w.
- M **acepta** w**:** $L(M_w) = \{w\} \neq \emptyset$, R rechaza, S acepta.
- M **no acepta** w**:** $L(M_w) = \emptyset$, R acepta, S rechaza.

E_{TM} es indecidible

Reducción desde A_{TM} hacia E_{TM}. Supongamos que R decide E_{TM}.

> S = **"Con la entrada $\langle M, w \rangle$:**
>
> 1. Construir la descripción de M_w.
> 2. Correr R con $\langle M_w \rangle$. Si R acepta, *rechazar*. Si R rechaza, *aceptar*."

- **Se detiene siempre:** escribe un texto y corre un decididor. No corre M ni M_w.
- M **acepta** w**:** $L(M_w) = \{w\} \neq \emptyset$**,** R rechaza, S acepta.
- M **no acepta** w**:** $L(M_w) = \emptyset$**,** R acepta, S rechaza.

Error común: escribir en S "correr M_w con todas las cadenas". No termina. S solo le pasa a R la *descripción* de M_w.

SECCIÓN 04

REGULAR y EQ

04

$\text{REGULAR}_{\text{TM}}$: ¿el lenguaje de una máquina es regular?

Dos lenguajes posibles, uno regular y otro no. M con w elige cuál.

> M_2 = **"Con la entrada x:**
>
> 1. Si x tiene la forma 0^n1^n, *aceptar*.
> 2. Si no, correr M con w. Si M acepta, *aceptar*."

$\text{REGULAR}_{\text{TM}}$: ¿el lenguaje de una máquina es regular?

Dos lenguajes posibles, uno regular y otro no. M con w elige cuál.

M_2 = "Con la entrada x:

1. Si x tiene la forma 0^n1^n, *aceptar*.
2. Si no, correr M con w. Si M acepta, *aceptar*."

$$L(M_2) = \Sigma^* \text{ si } M \text{ acepta } w, \qquad L(M_2) = \{0^n1^n\} \text{ si no.}$$

$\mathrm{REGULAR}_{\mathrm{TM}}$: ¿el lenguaje de una máquina es regular?

Dos lenguajes posibles, uno regular y otro no. M con w elige cuál.

> **M_2 = "Con la entrada x:**
>
> 1. Si x tiene la forma 0^n1^n, *aceptar*.
> 2. Si no, correr M con w. Si M acepta, *aceptar*."

$$L(M_2) = \Sigma^* \text{ si } M \text{ acepta } w, \qquad L(M_2) = \{0^n1^n\} \text{ si no.}$$

- **Desde A_{TM} hacia $\mathrm{REGULAR}_{\mathrm{TM}}$.** Si R decide $\mathrm{REGULAR}_{\mathrm{TM}}$, S construye $\langle M_2 \rangle$, corre R y responde **lo mismo**.

$\mathrm{REGULAR}_{\mathrm{TM}}$: ¿el lenguaje de una máquina es regular?

Dos lenguajes posibles, uno regular y otro no. M con w elige cuál.

> **M_2 = "Con la entrada x:**
>
> 1. Si x tiene la forma 0^n1^n, *aceptar*.
> 2. Si no, correr M con w. Si M acepta, *aceptar*."

$$L(M_2) = \Sigma^* \text{ si } M \text{ acepta } w, \qquad L(M_2) = \{0^n1^n\} \text{ si no.}$$

- **Desde A_{TM} hacia $\mathrm{REGULAR}_{\mathrm{TM}}$.** Si R decide $\mathrm{REGULAR}_{\mathrm{TM}}$, S construye $\langle M_2 \rangle$, corre R y responde **lo mismo**.
- M acepta w: Σ^* es regular, S acepta. M no acepta w: $\{0^n1^n\}$ no es regular (semana 4), S rechaza.

$\mathrm{REGULAR}_{\mathrm{TM}}$: ¿el lenguaje de una máquina es regular?

Dos lenguajes posibles, uno regular y otro no. M con w elige cuál.

> **M_2 = "Con la entrada x:**
>
> 1. Si x tiene la forma 0^n1^n, *aceptar*.
> 2. Si no, correr M con w. Si M acepta, *aceptar*."

$$L(M_2) = \Sigma^* \text{ si } M \text{ acepta } w, \qquad L(M_2) = \{0^n1^n\} \text{ si no.}$$

- **Desde A_{TM} hacia $\mathrm{REGULAR}_{\mathrm{TM}}$.** Si R decide $\mathrm{REGULAR}_{\mathrm{TM}}$, S construye $\langle M_2 \rangle$, corre R y responde **lo mismo**.
- M acepta w: Σ^* es regular, S acepta. M no acepta w: $\{0^n1^n\}$ no es regular (semana 4), S rechaza.
- M_2 nunca se ejecuta: solo importa su lenguaje.

$\mathrm{EQ}_{\mathrm{TM}}$: ¿dos máquinas reconocen lo mismo?

El problema

$\mathrm{EQ}_{\mathrm{TM}} = \{ \langle M_1, M_2 \rangle : L(M_1) = L(M_2) \}$

- Preguntar si $L(M) = \emptyset$ es preguntar si M equivale a una máquina que no acepta nada.

$\mathrm{EQ}_{\mathrm{TM}}$: ¿dos máquinas reconocen lo mismo?

El problema

$\mathrm{EQ}_{\mathrm{TM}} = \{ \langle M_1, M_2 \rangle : L(M_1) = L(M_2) \}$

- Preguntar si $L(M) = \emptyset$ es preguntar si M equivale a una máquina que no acepta nada.
- **Reducción desde E_{TM} hacia $\mathrm{EQ}_{\mathrm{TM}}$.** Sea $M_\emptyset$ una máquina que rechaza todo. Si R decide $\mathrm{EQ}_{\mathrm{TM}}$:

S = "Con la entrada $\langle M \rangle$:

1. Correr R con $\langle M, M_\emptyset \rangle$ y responder lo mismo."

$\mathrm{EQ}_{\mathrm{TM}}$: ¿dos máquinas reconocen lo mismo?

El problema

$\mathrm{EQ}_{\mathrm{TM}} = \{ \langle M_1, M_2 \rangle : L(M_1) = L(M_2) \}$

- Preguntar si $L(M) = \emptyset$ es preguntar si M equivale a una máquina que no acepta nada.
- **Reducción desde E_{TM} hacia $\mathrm{EQ}_{\mathrm{TM}}$.** Sea $M_\emptyset$ una máquina que rechaza todo. Si R decide $\mathrm{EQ}_{\mathrm{TM}}$:

S = "Con la entrada $\langle M \rangle$:

1. Correr R con $\langle M, M_\emptyset \rangle$ y responder lo mismo."

- $L(M) = \emptyset \iff L(M) = L(M_\emptyset)$. S decidiría E_{TM}: contradicción. Las reducciones se encadenan.

SECCIÓN 05

Reducción por función

05

Reducción por función

Función computable ***(computable function)***

$f : \Sigma^* \to \Sigma^*$ tal que alguna máquina, con cada w, **se detiene** dejando $f(w)$ en la cinta.

Reducción por función

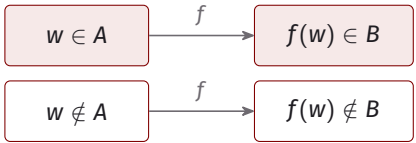
Función computable *(computable function)*

$f : \Sigma^* \to \Sigma^*$ tal que alguna máquina, con cada w, **se detiene** dejando $f(w)$ en la cinta.

$A \leq_m B$ *(mapping reducible)*

Existe f computable tal que, para toda w:

$$w \in A \iff f(w) \in B.$$



Reducción por función

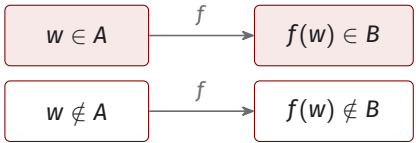
Función computable *(computable function)*

$f : \Sigma^* \to \Sigma^*$ tal que alguna máquina, con cada w, **se detiene** dejando $f(w)$ en la cinta.

$A \leq_m B$ *(mapping reducible)*

Existe f computable tal que, para toda w:

$$w \in A \iff f(w) \in B.$$



- Son **dos** condiciones: lo de A cae dentro de B, y lo que no está en A cae fuera.

Reducción por función

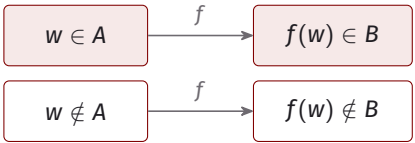
Función computable *(computable function)*

$f\colon \Sigma^* \to \Sigma^*$ tal que alguna máquina, con cada w, **se detiene** dejando $f(w)$ en la cinta.

$A \leq_m B$ *(mapping reducible)*

Existe f computable tal que, para toda w:

$$w \in A \iff f(w) \in B.$$



- Son **dos** condiciones: lo de A cae dentro de B, y lo que no está en A cae fuera.
- `fix_input` calcula $\langle M, w\rangle \mapsto \langle M_w\rangle$: transformar descripciones es computable.

Qué conserva

Si $A \leq_m B$ **y...** **entonces...**

N = "Con la entrada w:

1. Calcular $f(w)$.
2. Correr M con $f(w)$. Responder lo mismo."

Qué conserva

N = "Con la entrada w:

1. Calcular $f(w)$.
2. Correr M con $f(w)$. Responder lo mismo."

- M decide B: N se detiene siempre, y acepta $w \iff f(w) \in B \iff w \in A$.

Si $A \leq_m B$ **y...**	**entonces...**
B es decidible	A es decidible

Qué conserva

N = "Con la entrada w:

1. Calcular $f(w)$.
2. Correr M con $f(w)$. Responder lo mismo."

- M decide B: N se detiene siempre, y acepta $w \iff f(w) \in B \iff w \in A$.
- M reconoce B: si $f(w) \notin B$, M no acepta, y N tampoco.

Si $A \leq_m B$ y...	**entonces...**
B es decidible	A es decidible
B es reconocible	A es reconocible

Qué conserva

N = "Con la entrada w:

1. Calcular $f(w)$.
2. Correr M con $f(w)$. Responder lo mismo."

- M decide B: N se detiene siempre, y acepta $w \iff f(w) \in B \iff w \in A$.
- M reconoce B: si $f(w) \notin B$, M no acepta, y N tampoco.

Si $A \leq_m B$ y...	**entonces...**
B es decidible	A es decidible
A es indecidible	B es indecidible
B es reconocible	A es reconocible
A no es reconocible	B no es reconocible

Qué conserva

N = "Con la entrada w:

1. Calcular $f(w)$.
2. Correr M con $f(w)$. Responder lo mismo."

Si $A \leq_m B$ **y...**	**entonces...**
B es decidible	A es decidible
A es indecidible	B es indecidible
B es reconocible	A es reconocible
A no es reconocible	B no es reconocible

- M decide B: N se detiene siempre, y acepta $w \iff f(w) \in B \iff w \in A$.
- M reconoce B: si $f(w) \notin B$, M no acepta, y N tampoco.

Para no equivocar la dirección: lo bueno baja de B hacia A; lo malo sube de A hacia B.

Complementos, y las reducciones de hoy como funciones

- **Complementos:** $A \leq_m B \iff \overline{A} \leq_m \overline{B}$, con la misma f. $\quad (w \notin A \iff f(w) \notin B)$
- **Transitividad:** si $A \leq_m B$ y $B \leq_m C$, entonces $A \leq_m C$, con $g \circ f$.

Complementos, y las reducciones de hoy como funciones

- **Complementos:** $A \leq_m B \iff \overline{A} \leq_m \overline{B}$, con la misma f. $(w \notin A \iff f(w) \notin B)$
- **Transitividad:** si $A \leq_m B$ y $B \leq_m C$, entonces $A \leq_m C$, con $g \circ f$.

Reducción	La función	Por qué
$A_{\mathrm{TM}} \leq_m \mathrm{HALT}_{\mathrm{TM}}$	$\langle M, w\rangle \mapsto \langle M', w\rangle$	M': como M, pero en vez de rechazar no se detiene
$A_{\mathrm{TM}} \leq_m \mathrm{REGULAR}_{\mathrm{TM}}$	$\langle M, w\rangle \mapsto \langle M_2\rangle$	S respondía lo mismo que R
$E_{\mathrm{TM}} \leq_m \mathrm{EQ}_{\mathrm{TM}}$	$\langle M\rangle \mapsto \langle M, M_\emptyset\rangle$	
$A_{\mathrm{TM}} \leq_m \overline{E_{\mathrm{TM}}}$	$\langle M, w\rangle \mapsto \langle M_w\rangle$	S respondía lo contrario de R

Complementos, y las reducciones de hoy como funciones

- **Complementos:** $A \leq_m B \iff \overline{A} \leq_m \overline{B}$, con la misma f. $\quad (w \notin A \iff f(w) \notin B)$
- **Transitividad:** si $A \leq_m B$ y $B \leq_m C$, entonces $A \leq_m C$, con $g \circ f$.

Reducción	La función	Por qué
$A_{\mathrm{TM}} \leq_m \mathrm{HALT}_{\mathrm{TM}}$	$\langle M, w\rangle \mapsto \langle M', w\rangle$	M': como M, pero en vez de rechazar no se detiene
$A_{\mathrm{TM}} \leq_m \mathrm{REGULAR}_{\mathrm{TM}}$	$\langle M, w\rangle \mapsto \langle M_2\rangle$	S respondía lo mismo que R
$E_{\mathrm{TM}} \leq_m \mathrm{EQ}_{\mathrm{TM}}$	$\langle M\rangle \mapsto \langle M, M_\emptyset\rangle$	
$A_{\mathrm{TM}} \leq_m \overline{E_{\mathrm{TM}}}$	$\langle M, w\rangle \mapsto \langle M_w\rangle$	S respondía lo contrario de R

La diferencia

Usar un decididor como subrutina permite invertir la respuesta. La reducción por función, no. Para lo indecidible da igual; para lo reconocible, es todo.

E_{TM} no es reconocible

- $\overline{E_{\mathrm{TM}}}$ **es** reconocible: buscar una cadena que M acepte, en paralelo. Para $i = 1, 2, 3, \ldots$: correr M durante i pasos con cada una de las primeras i cadenas.

E_{TM} no es reconocible

- $\overline{E_{\mathrm{TM}}}$ **es** reconocible: buscar una cadena que M acepte, en paralelo. Para $i = 1, 2, 3, \ldots$: correr M durante i pasos con cada una de las primeras i cadenas.
- De $A_{\mathrm{TM}} \leq_m \overline{E_{\mathrm{TM}}}$ sale $\overline{A_{\mathrm{TM}}} \leq_m E_{\mathrm{TM}}$. Y $\overline{A_{\mathrm{TM}}}$ no es reconocible (semana 9).

Corolario

E_{TM} no es reconocible.

E_{TM} no es reconocible

- $\overline{E_{\mathrm{TM}}}$ **es** reconocible: buscar una cadena que M acepte, en paralelo. Para $i = 1, 2, 3, \ldots$: correr M durante i pasos con cada una de las primeras i cadenas.
- De $A_{\mathrm{TM}} \leq_m \overline{E_{\mathrm{TM}}}$ sale $\overline{A_{\mathrm{TM}}} \leq_m E_{\mathrm{TM}}$. Y $\overline{A_{\mathrm{TM}}}$ no es reconocible (semana 9).

Corolario

E_{TM} no es reconocible.

- ¿Y una función más ingeniosa que dé $A_{\mathrm{TM}} \leq_m E_{\mathrm{TM}}$? **No existe**: daría $\overline{A_{\mathrm{TM}}} \leq_m \overline{E_{\mathrm{TM}}}$, y $\overline{A_{\mathrm{TM}}}$ sería reconocible.

E_{TM} no es reconocible

- $\overline{E_{\mathrm{TM}}}$ **es** reconocible: buscar una cadena que M acepte, en paralelo. Para $i = 1, 2, 3, \ldots$: correr M durante i pasos con cada una de las primeras i cadenas.
- De $A_{\mathrm{TM}} \leq_m \overline{E_{\mathrm{TM}}}$ sale $\overline{A_{\mathrm{TM}}} \leq_m E_{\mathrm{TM}}$. Y $\overline{A_{\mathrm{TM}}}$ no es reconocible (semana 9).

Corolario

E_{TM} no es reconocible.

- ¿Y una función más ingeniosa que dé $A_{\mathrm{TM}} \leq_m E_{\mathrm{TM}}$? **No existe**: daría $\overline{A_{\mathrm{TM}}} \leq_m \overline{E_{\mathrm{TM}}}$, y $\overline{A_{\mathrm{TM}}}$ sería reconocible.
- Para convencerse de que M acepta algo basta encontrar una cadena. Para convencerse de que no acepta *nada*, ninguna búsqueda finita alcanza.

$\mathrm{EQ}_{\mathrm{TM}}$ no es reconocible, y su complemento tampoco

Dos reducciones desde $\overline{A_{\mathrm{TM}}}$, con la misma máquina auxiliar:

M_2 = "Con cualquier entrada x:

1. Correr M con w. Si M acepta, *aceptar*."

$L(M_2) = \Sigma^*$ si M acepta w; $\emptyset$ si no.

M_1	**Si M acepta w**	**Se obtiene**	**Conclusión**

$\mathrm{EQ}_{\mathrm{TM}}$ no es reconocible, y su complemento tampoco

Dos reducciones desde $\overline{A_{\mathrm{TM}}}$, con la misma máquina auxiliar:

M_2 = "Con cualquier entrada x:

1. Correr M con w. Si M acepta, *aceptar*."

$L(M_2) = \Sigma^*$ si M acepta w; $\emptyset$ si no.

	M_1	**Si M acepta w**	**Se obtiene**	**Conclusión**
1	rechaza todo	$L(M_1) \neq L(M_2)$	$A_{\mathrm{TM}} \leq_m \overline{\mathrm{EQ}_{\mathrm{TM}}}$	$\mathrm{EQ}_{\mathrm{TM}}$ no es reconocible

$\mathrm{EQ}_{\mathrm{TM}}$ no es reconocible, y su complemento tampoco

Dos reducciones desde $\overline{A_{\mathrm{TM}}}$, con la misma máquina auxiliar:

M_2 = "Con cualquier entrada x:

1. Correr M con w. Si M acepta, *aceptar*." $L(M_2) = \Sigma^*$ si M acepta w; $\emptyset$ si no.

	M_1	**Si M acepta w**	**Se obtiene**	**Conclusión**
1	rechaza todo	$L(M_1) \neq L(M_2)$	$A_{\mathrm{TM}} \leq_m \overline{\mathrm{EQ}_{\mathrm{TM}}}$	$\mathrm{EQ}_{\mathrm{TM}}$ no es reconocible
2	acepta todo	$L(M_1) = L(M_2)$	$A_{\mathrm{TM}} \leq_m \mathrm{EQ}_{\mathrm{TM}}$	$\overline{\mathrm{EQ}_{\mathrm{TM}}}$ no es reconocible

$\mathrm{EQ}_{\mathrm{TM}}$ no es reconocible, y su complemento tampoco

Dos reducciones desde $\overline{A_{\mathrm{TM}}}$, con la misma máquina auxiliar:

M_2 = "Con cualquier entrada x:

1. Correr M con w. Si M acepta, *aceptar*." $L(M_2) = \Sigma^*$ si M acepta w; $\emptyset$ si no.

	M_1	**Si M acepta w**	**Se obtiene**	**Conclusión**
1	rechaza todo	$L(M_1) \neq L(M_2)$	$A_{\mathrm{TM}} \leq_m \overline{\mathrm{EQ}_{\mathrm{TM}}}$	$\mathrm{EQ}_{\mathrm{TM}}$ no es reconocible
2	acepta todo	$L(M_1) = L(M_2)$	$A_{\mathrm{TM}} \leq_m \mathrm{EQ}_{\mathrm{TM}}$	$\overline{\mathrm{EQ}_{\mathrm{TM}}}$ no es reconocible

Cómo leer la tabla

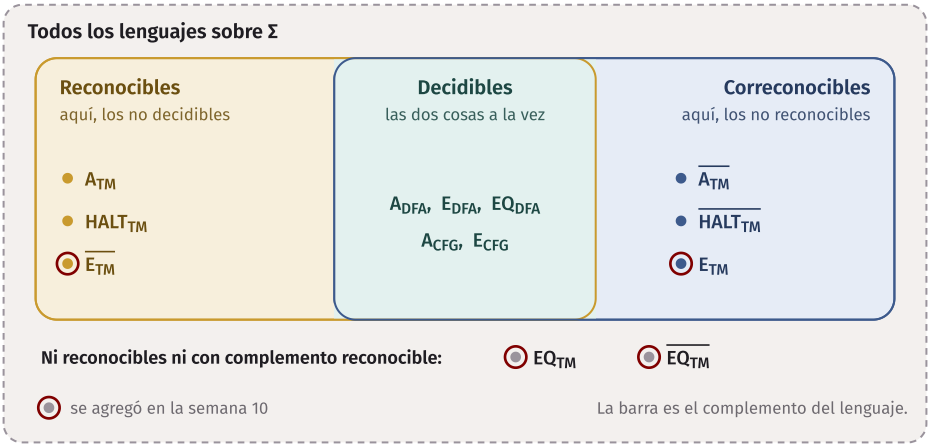
Si $A_{\mathrm{TM}} \leq_m X$, entonces $\overline{A_{\mathrm{TM}}} \leq_m \overline{X}$, y el que queda sin reconocedor es $\overline{X}$.

El mapa, con lo de hoy

Región	Ejemplos
Decidibles	A_{DFA}, E_{DFA}, $\mathrm{EQ}_{\mathrm{DFA}}$, A_{CFG}, E_{CFG}
Reconocibles, no decidibles	A_{TM}, $\mathrm{HALT}_{\mathrm{TM}}$, $\overline{E_{\mathrm{TM}}}$
No reconocibles, con complemento reconocible	$\overline{A_{\mathrm{TM}}}$, $\overline{\mathrm{HALT}_{\mathrm{TM}}}$, E_{TM}
Ni reconocibles ni con complemento reconocible	$\mathrm{EQ}_{\mathrm{TM}}$, $\overline{\mathrm{EQ}_{\mathrm{TM}}}$

- En la última región, ni el “sí” ni el “no” se pueden confirmar con una búsqueda.

El mapa, dibujado



SECCIÓN 06

El teorema de Rice

06



El teorema de Rice

Sea P un lenguaje de descripciones de máquinas de Turing

- P es una **propiedad del lenguaje** si, siempre que $L(M_1) = L(M_2)$: $\langle M_1 \rangle \in P \iff \langle M_2 \rangle \in P$.
- P es **no trivial** si alguna máquina la tiene y alguna no.

El teorema de Rice

Sea P un lenguaje de descripciones de máquinas de Turing

- P es una **propiedad del lenguaje** si, siempre que $L(M_1) = L(M_2)$: $\langle M_1 \rangle \in P \iff \langle M_2 \rangle \in P$.
- P es **no trivial** si alguna máquina la tiene y alguna no.

Teorema de Rice

Toda propiedad no trivial del lenguaje de una máquina de Turing es indecidible.

El teorema de Rice

Sea P un lenguaje de descripciones de máquinas de Turing

- P es una **propiedad del lenguaje** si, siempre que $L(M_1) = L(M_2)$: $\langle M_1 \rangle \in P \iff \langle M_2 \rangle \in P$.
- P es **no trivial** si alguna máquina la tiene y alguna no.

Teorema de Rice

Toda propiedad no trivial del lenguaje de una máquina de Turing es indecidible.

- P no ve los estados, ni los pasos, ni cómo está escrita la tabla. Y no es una pregunta que se responda sin mirar.

Demostración: la construcción

Reducción desde A_{TM} hacia P. Supongamos que R_P decide P.

1. Sea $T_\emptyset$ una máquina que rechaza todo. Se puede suponer $\langle T_\emptyset \rangle \notin P$: si no, se trabaja con $\overline{P}$, que también es propiedad del lenguaje, no trivial, y decidible si P lo es.

Demostración: la construcción

Reducción desde A_{TM} hacia P. Supongamos que R_P decide P.

1. Sea $T_\emptyset$ una máquina que rechaza todo. Se puede suponer $\langle T_\emptyset \rangle \notin P$: si no, se trabaja con $\overline{P}$, que también es propiedad del lenguaje, no trivial, y decidible si P lo es.
2. P es no trivial: existe T con $\langle T \rangle \in P$.

Demostración: la construcción

Reducción desde A_{TM} hacia P. Supongamos que R_P decide P.

1. Sea $T_\emptyset$ una máquina que rechaza todo. Se puede suponer $\langle T_\emptyset \rangle \notin P$: si no, se trabaja con $\overline{P}$, que también es propiedad del lenguaje, no trivial, y decidible si P lo es.
2. P es no trivial: existe T con $\langle T \rangle \in P$.
3. Dados M y w, se construye:

> **M_w = "Con la entrada x:**
>
> 1. Correr M con w. Si M rechaza, *rechazar*.
> 2. Si M acepta, correr T con x. Si T acepta, *aceptar*."

Demostración: la construcción

Reducción desde A_{TM} hacia P. Supongamos que R_P decide P.

1. Sea $T_\emptyset$ una máquina que rechaza todo. Se puede suponer $\langle T_\emptyset \rangle \notin P$: si no, se trabaja con $\overline{P}$, que también es propiedad del lenguaje, no trivial, y decidible si P lo es.
2. P es no trivial: existe T con $\langle T \rangle \in P$.
3. Dados M y w, se construye:

> **M_w = "Con la entrada x:**
>
> 1. Correr M con w. Si M rechaza, *rechazar*.
> 2. Si M acepta, correr T con x. Si T acepta, *aceptar*."

$$L(M_w) = L(T) \text{ si } M \text{ acepta } w, \qquad L(M_w) = \emptyset \text{ si no.}$$

Demostración: los dos casos

S = "Con la entrada $\langle M, w \rangle$:

1. Construir $\langle M_w \rangle$.
2. Correr R_P con $\langle M_w \rangle$ y responder lo mismo."

Demostración: los dos casos

> S = **"Con la entrada** $\langle M, w \rangle$**:**
>
> 1. Construir $\langle M_w \rangle$.
> 2. Correr R_P con $\langle M_w \rangle$ y responder lo mismo."

- M **acepta** w**:** $L(M_w) = L(T)$ y $\langle T \rangle \in P$. Como P solo depende del lenguaje, $\langle M_w \rangle \in P$. S acepta.

Demostración: los dos casos

S = "Con la entrada $\langle M, w \rangle$:

1. Construir $\langle M_w \rangle$.
2. Correr R_P con $\langle M_w \rangle$ y responder lo mismo."

- M **acepta** w**:** $L(M_w) = L(T)$ y $\langle T \rangle \in P$. Como P solo depende del lenguaje, $\langle M_w \rangle \in P$. S acepta.
- M **no acepta** w**:** $L(M_w) = \emptyset = L(T_\emptyset)$ y $\langle T_\emptyset \rangle \notin P$. Entonces $\langle M_w \rangle \notin P$. S rechaza.

Demostración: los dos casos

S = "Con la entrada $\langle M, w \rangle$:

1. Construir $\langle M_w \rangle$.
2. Correr R_P con $\langle M_w \rangle$ y responder lo mismo."

- M **acepta** w**:** $L(M_w) = L(T)$ y $\langle T \rangle \in P$. Como P solo depende del lenguaje, $\langle M_w \rangle \in P$. S acepta.
- M **no acepta** w**:** $L(M_w) = \emptyset = L(T_\emptyset)$ y $\langle T_\emptyset \rangle \notin P$. Entonces $\langle M_w \rangle \notin P$. S rechaza.
- S decidiría A_{TM}. Contradicción.

Demostración: los dos casos

S = "Con la entrada $\langle M, w \rangle$:

1. Construir $\langle M_w \rangle$.
2. Correr R_P con $\langle M_w \rangle$ y responder lo mismo."

- M **acepta** w**:** $L(M_w) = L(T)$ y $\langle T \rangle \in P$. Como P solo depende del lenguaje, $\langle M_w \rangle \in P$. S acepta.
- M **no acepta** w**:** $L(M_w) = \emptyset = L(T_\emptyset)$ y $\langle T_\emptyset \rangle \notin P$. Entonces $\langle M_w \rangle \notin P$. S rechaza.
- S decidiría A_{TM}. Contradicción.

Dónde se usa cada hipótesis

No trivial da las dos máquinas, T y $T_\emptyset$, una a cada lado. **Propiedad del lenguaje** permite pasar de "M_w reconoce lo mismo que T" a "M_w tiene la propiedad".

Cómo se usa

Tres cosas que hay que escribir: que solo depende de $L(M)$, una máquina que la tiene y una que no.

Propiedad	**¿Solo $L(M)$?**	**Una que la tiene**	**Una que no**
$L(M) = \emptyset$	sí	la que rechaza todo	la que acepta todo
$L(M)$ es regular	sí	la que acepta todo	la que decide $\{0^n 1^n\}$
$L(M)$ es finito	sí	la que rechaza todo	la que acepta todo
$\varepsilon \in L(M)$	sí	la que acepta todo	la que rechaza todo
$L(M) = \Sigma^*$	sí	la que acepta todo	la que rechaza todo
$L(M)$ es libre de contexto	sí	la que rechaza todo	la que decide $\{a^n b^n c^n\}$

Cómo se usa

Tres cosas que hay que escribir: que solo depende de $L(M)$, una máquina que la tiene y una que no.

Propiedad	**¿Solo $L(M)$?**	**Una que la tiene**	**Una que no**
$L(M) = \emptyset$	sí	la que rechaza todo	la que acepta todo
$L(M)$ es regular	sí	la que acepta todo	la que decide $\{0^n 1^n\}$
$L(M)$ es finito	sí	la que rechaza todo	la que acepta todo
$\varepsilon \in L(M)$	sí	la que acepta todo	la que rechaza todo
$L(M) = \Sigma^*$	sí	la que acepta todo	la que rechaza todo
$L(M)$ es libre de contexto	sí	la que rechaza todo	la que decide $\{a^n b^n c^n\}$

- "M acepta ε" parece hablar de la máquina, pero es "$\varepsilon \in L(M)$".

Cómo se usa

Tres cosas que hay que escribir: que solo depende de $L(M)$, una máquina que la tiene y una que no.

Propiedad	**¿Solo $L(M)$?**	**Una que la tiene**	**Una que no**
$L(M) = \emptyset$	sí	la que rechaza todo	la que acepta todo
$L(M)$ es regular	sí	la que acepta todo	la que decide $\{0^n 1^n\}$
$L(M)$ es finito	sí	la que rechaza todo	la que acepta todo
$\varepsilon \in L(M)$	sí	la que acepta todo	la que rechaza todo
$L(M) = \Sigma^*$	sí	la que acepta todo	la que rechaza todo
$L(M)$ es libre de contexto	sí	la que rechaza todo	la que decide $\{a^n b^n c^n\}$

- "M acepta ε" parece hablar de la máquina, pero es "$\varepsilon \in L(M)$".
- En la práctica: ningún analizador estático es exacto. Rechaza programas buenos, deja pasar malos, o dice "no sé".

Lo que Rice no dice: propiedades de la máquina

- **"M tiene 5 estados."** Agregar un estado sin usar no cambia $L(M)$: no es propiedad del lenguaje. Y es decidible: se cuenta.

Lo que Rice no dice: propiedades de la máquina

- **"M tiene 5 estados."** Agregar un estado sin usar no cambia $L(M)$: no es propiedad del lenguaje. Y es decidible: se cuenta.
- **"M se detiene con w en a lo sumo 100 pasos."** Decidible: simular *con tope* siempre termina.

Lo que Rice no dice: propiedades de la máquina

- **"M tiene 5 estados."** Agregar un estado sin usar no cambia $L(M)$: no es propiedad del lenguaje. Y es decidible: se cuenta.
- **"M se detiene con w en a lo sumo 100 pasos."** Decidible: simular *con tope* siempre termina.
- **"M se detiene con toda entrada en a lo sumo 100 pasos."** En 100 pasos el cabezal solo lee 100 celdas: basta probar las entradas de longitud ≤ 100. Decidible.

Lo que Rice no dice: propiedades de la máquina

- **"M tiene 5 estados."** Agregar un estado sin usar no cambia $L(M)$: no es propiedad del lenguaje. Y es decidible: se cuenta.
- **"M se detiene con w en a lo sumo 100 pasos."** Decidible: simular *con tope* siempre termina.
- **"M se detiene con toda entrada en a lo sumo 100 pasos."** En 100 pasos el cabezal solo lee 100 celdas: basta probar las entradas de longitud ≤ 100. Decidible.

La prueba antes de usar Rice

Si agregar un estado o hacer más lenta la máquina cambia la respuesta sin cambiar $L(M)$, la propiedad es de la máquina, y el teorema no se aplica.

Simular con tope siempre termina

terminal

```
$ build/history machines/loop_on_0.tm 00 --pasos 5
C1   q0 0 0
C2   0 q1 0
C3   0 0 q1 _
C4   0 0 _ q1 _
C5   0 0 _ _ q1 _
C6   0 0 _ _ _ q1 _
sin detenerse después de 5 pasos: no hay historia que mostrar
```

- La respuesta es exacta: en 5 pasos no se detuvo.
- Lo que ningún programa puede responder es si se detendrá **alguna vez**.

Que Rice no se aplique no quiere decir “decidible”

Propiedad	¿Cae bajo Rice?	¿Decidible?
$L(M)$ es finito	sí: del lenguaje, no trivial	no, por Rice
$L(M)$ contiene a $\odot$	sí: del lenguaje, no trivial	no, por Rice
M se detiene siempre en menos de 10 pasos	no: es de la máquina	**sí**: entradas cortas, con tope
M se detiene con la entrada $\odot$	no: es de la máquina	no, por otra reducción
M tiene un estado inútil	no: es de la máquina	no, por otra reducción
$L(M)$ es reconocible	no: es trivial	**sí**: aceptar siempre

Que Rice no se aplique no quiere decir “decidible”

Propiedad	¿Cae bajo Rice?	¿Decidible?
$L(M)$ es finito	sí: del lenguaje, no trivial	no, por Rice
$L(M)$ contiene a $\odot$	sí: del lenguaje, no trivial	no, por Rice
M se detiene siempre en menos de 10 pasos	no: es de la máquina	**sí**: entradas cortas, con tope
M se detiene con la entrada $\odot$	no: es de la máquina	no, por otra reducción
M tiene un estado inútil	no: es de la máquina	no, por otra reducción
$L(M)$ es reconocible	no: es trivial	**sí**: aceptar siempre

Dos errores comunes

Aplicar Rice sin escribir la propiedad en términos de $L(M)$ ni dar las dos máquinas. Y reducir al revés: $P \leq_m A_{\mathrm{TM}}$ vale para todo lenguaje reconocible.

SECCIÓN 07

Historias de cómputo

07

Problemas que no hablan de máquinas

- Todo lo indecidible de la sesión A tiene una máquina de Turing en la entrada.

Problemas que no hablan de máquinas

- Todo lo indecidible de la sesión A tiene una máquina de Turing en la entrada.
- ¿Están a salvo los problemas de cadenas, fichas o gramáticas? No.

Problemas que no hablan de máquinas

- Todo lo indecidible de la sesión A tiene una máquina de Turing en la entrada.
- ¿Están a salvo los problemas de cadenas, fichas o gramáticas? No.

Hoy, tres problemas sin máquinas en la entrada:

1. un rompecabezas de fichas: el problema de correspondencia de Post;
2. ¿una gramática libre de contexto genera *todas* las cadenas?
3. ¿dos gramáticas generan el mismo lenguaje? (la deuda de la semana 9)

Problemas que no hablan de máquinas

- Todo lo indecidible de la sesión A tiene una máquina de Turing en la entrada.
- ¿Están a salvo los problemas de cadenas, fichas o gramáticas? No.

Hoy, tres problemas sin máquinas en la entrada:

1. un rompecabezas de fichas: el problema de correspondencia de Post;
2. ¿una gramática libre de contexto genera *todas* las cadenas?
3. ¿dos gramáticas generan el mismo lenguaje? (la deuda de la semana 9)

Hace falta un puente

La reducción tiene que construir fichas, o una gramática, a partir de $\langle M, w \rangle$. Las fichas no ejecutan nada. El puente es escribir la ejecución entera como una cadena.

Historia de cómputo

Historia de cómputo de aceptación *(accepting computation history)*

Una secuencia de configuraciones $C_1, C_2, \ldots, C_k$ de M con la entrada w tal que:

1. C_1 es la configuración inicial, $q_0\, w$;
2. cada C_i produce C_{i+1};
3. C_k tiene a q_{accept}.

Se escribe como una sola cadena: $C_1 \# C_2 \# \cdots \# C_k$.

Historia de cómputo

Historia de cómputo de aceptación *(accepting computation history)*

Una secuencia de configuraciones $C_1, C_2, \ldots, C_k$ de M con la entrada w tal que:

1. C_1 es la configuración inicial, $q_0\, w$;
2. cada C_i produce C_{i+1};
3. C_k tiene a q_{accept}.

Se escribe como una sola cadena: $C_1 \# C_2 \# \cdots \# C_k$.

- Es **finita**. Si M no se detiene con w, no hay historia.

Historia de cómputo

Historia de cómputo de aceptación *(accepting computation history)*

Una secuencia de configuraciones $C_1, C_2, \ldots, C_k$ de M con la entrada w tal que:

1. C_1 es la configuración inicial, $q_0\, w$;
2. cada C_i produce C_{i+1};
3. C_k tiene a q_{accept}.

Se escribe como una sola cadena: $C_1 \# C_2 \# \cdots \# C_k$.

- Es **finita**. Si M no se detiene con w, no hay historia.
- M acepta $w \iff$ existe una historia de aceptación de M con w.

Historia de cómputo

Historia de cómputo de aceptación *(accepting computation history)*

Una secuencia de configuraciones $C_1, C_2, \ldots, C_k$ de M con la entrada w tal que:

1. C_1 es la configuración inicial, $q_0\, w$;
2. cada C_i produce C_{i+1};
3. C_k tiene a q_{accept}.

Se escribe como una sola cadena: $C_1 \# C_2 \# \cdots \# C_k$.

- Es **finita**. Si M no se detiene con w, no hay historia.
- M acepta $w \iff$ existe una historia de aceptación de M con w.
- M es determinista: si existe, es una sola (salvo los blancos que se anoten al final).

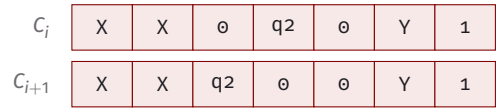
Una historia, escrita por un programa

terminal

```
$ build/history machines/zeros_ones.tm 01
C1   q0 0 1
C2   X q1 1
C3   q2 X Y
C4   X q0 Y
C5   X Y q3 _
C6   X Y _ q_accept _
acepta después de 5 pasos
historia: q0 0 1 # X q1 1 # q2 X Y # X q0 Y # X Y q3 _ # X Y _ q_accept _
```

- La máquina decide $\{0^n 1^n\}$ tachando un `0` con `X` y un `1` con `Y`.
- `q_accept` es *un* símbolo de la historia. El alfabeto de una historia es $\Gamma \cup Q \cup \{\#\}$.
- Una historia es una cadena: se puede preguntar si una cadena dada es o no una historia.

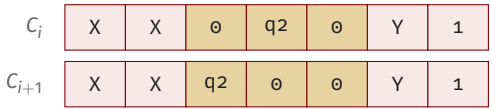
Comprobar una historia es local



$\delta(q_2, \texttt{o}) = (q_2, \texttt{o}, \mathrm{L})$

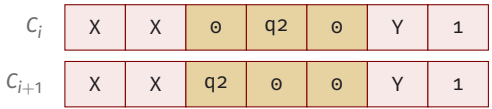
- En un paso cambia el estado, se escribe **una** celda y el cabezal se mueve **una** posición.

Comprobar una historia es local



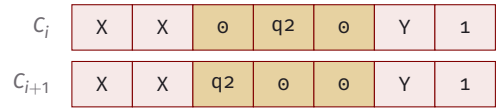
- En un paso cambia el estado, se escribe **una** celda y el cabezal se mueve **una** posición.
- Solo pueden cambiar **tres símbolos**, el estado y sus dos vecinos: $c\,q\,a \to c\,b\,r$ (R) o $c\,q\,a \to r\,c\,b$ (L).

Comprobar una historia es local



- En un paso cambia el estado, se escribe **una** celda y el cabezal se mueve **una** posición.
- Solo pueden cambiar **tres símbolos**, el estado y sus dos vecinos: $c\,q\,a \to c\,b\,r$ (R) o $c\,q\,a \to r\,c\,b$ (L).
- Todo lo demás se copia igual.

Comprobar una historia es local



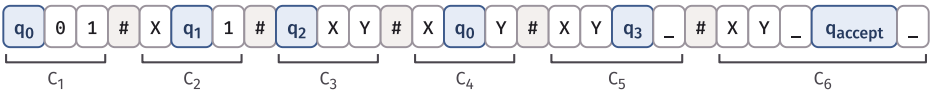
- En un paso cambia el estado, se escribe **una** celda y el cabezal se mueve **una** posición.
- Solo pueden cambiar **tres símbolos**, el estado y sus dos vecinos: $c\,q\,a \to c\,b\,r$ (R) o $c\,q\,a \to r\,c\,b$ (L).
- Todo lo demás se copia igual.

Revisar por ventanas de tres símbolos

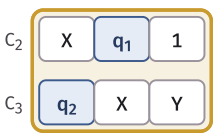
Sin estado en la ventana de arriba: el centro se copia. Con el estado en el centro: los tres de abajo los manda δ. No se simula nada ni se recuerda la cinta.

Las ventanas, en la historia de 01

La historia de `zeros_ones.tm` con `01`, como una sola cadena:

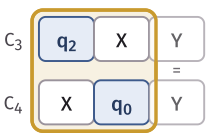


Dos configuraciones seguidas, una sobre otra: solo cambia la ventana del estado



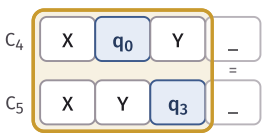
$\delta(q_1, 1) = (q_2, Y, L)$

a la izquierda: el estado salta sobre la X



$\delta(q_2, X) = (q_0, X, R)$

a la derecha, desde la primera celda



$\delta(q_0, Y) = (q_3, Y, R)$

a la derecha: la X de la ventana queda igual

Fuera de la ventana cada símbolo se copia igual: comprobar un paso es comparar símbolos vecinos.

El comprobador

terminal

```
$ build/check_history machines/starts_01.tm 01 "q0 0 1 # 0 q1 1 # q2 0 1 # 0 q_accept 1"
es una historia de aceptación
$ build/check_history machines/starts_01.tm 01 "q0 0 1 # 0 q1 1 # q2 0 0 # 0 q_accept 0"
no es una historia de aceptación
  de C2 a C3: ventana del cabezal en la posición 2: arriba [0 q1 1], abajo [q2 0 0], y δ pide [q2 0 1]
```

Revisa: cada tramo es una configuración; $C_1 = q_0\, w$; cada par pasa las ventanas; la última acepta.

La idea de toda la sesión

"¿M acepta w?" es indecidible. "¿Es esta cadena una historia de aceptación?" se responde comparando símbolos vecinos. Lo difícil es *encontrar* la historia, no revisarla.

SECCIÓN 08

El problema de correspondencia de Post

08

El problema de correspondencia de Post (PCP)

Instancia y solución

Fichas $\left[\frac{t_1}{b_1}\right], \ldots, \left[\frac{t_k}{b_k}\right]$. Una **solución** es una secuencia $i_1, \ldots, i_n$, con $n \geq 1$, tal que

$$t_{i_1} t_{i_2} \cdots t_{i_n} = b_{i_1} b_{i_2} \cdots b_{i_n}.$$

- Las fichas se repiten cuanto se quiera.
- No se voltean ni se parten.

Cuatro fichas:

b	a	ca	abc
ca	ab	a	c

El problema de correspondencia de Post (PCP)

Instancia y solución

Fichas $\left[\frac{t_1}{b_1}\right], \ldots, \left[\frac{t_k}{b_k}\right]$. Una **solución** es una secuencia $i_1, \ldots, i_n$, con $n \geq 1$, tal que

$$t_{i_1} t_{i_2} \cdots t_{i_n} = b_{i_1} b_{i_2} \cdots b_{i_n}.$$

- Las fichas se repiten cuanto se quiera.
- No se voltean ni se parten.

Cuatro fichas:

b / ca	a / ab	ca / a	abc / c

Una solución, 2 1 3 2 4:

a / ab	b / ca	ca / a	a / ab	abc / c

Arriba y abajo: abcaaabc

El problema de correspondencia de Post (PCP)

Instancia y solución

Fichas $\left[\frac{t_1}{b_1}\right], \ldots, \left[\frac{t_k}{b_k}\right]$. Una **solución** es una secuencia $i_1, \ldots, i_n$, con $n \geq 1$, tal que

$$t_{i_1} t_{i_2} \cdots t_{i_n} = b_{i_1} b_{i_2} \cdots b_{i_n}.$$

- Las fichas se repiten cuanto se quiera.
- No se voltean ni se parten.

Emil Post, 1946. $\mathrm{PCP} = \{ \langle P \rangle : P \text{ tiene una solución} \}$.

Cuatro fichas:

b / ca	a / ab	ca / a	abc / c

Una solución, 2 1 3 2 4:

a / ab	b / ca	ca / a	a / ab	abc / c

Arriba y abajo: abcaaabc

Seguir el exceso

```
terminal

$ build/pcp_check instances/sipser.pcp 2 1 3 2 4
arriba: a b ca a abc  =  abcaaabc
abajo:  ab ca a ab c  =  abcaaabc
es una solución (5 fichas, 8 símbolos)
$ build/pcp_check instances/sipser.pcp 2 1 3
arriba: a b ca  =  abca
abajo:  ab ca a  =  abcaa
no es una solución: abajo sobra a
```

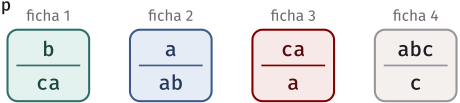
- Mientras coinciden, una cadena es prefijo de la otra.
- Solo importa **qué lado** va adelantado y **con qué cadena**: el exceso.

Ficha	Arriba	Abajo	Exceso
2	a	ab	abajo, b
1	ab	abca	abajo, ca
3	abca	abcaa	abajo, a
2	abcaa	abcaaab	abajo, ab
4	abcaaabc	abcaaabc	vacío

Una solución es un camino que sale del exceso vacío y vuelve a él.

La solución, ficha por ficha

Las fichas de `sipser.pcp`



La fila

arriba

abajo

Hay que poner fichas en fila hasta que arriba y abajo se lea lo mismo.

Solo la ficha 2 puede ir primera: es la única que empieza igual arriba y abajo.

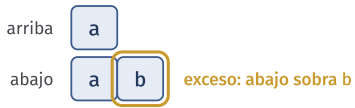
La solución, ficha por ficha

Las fichas de `sipser.pcp`



La fila

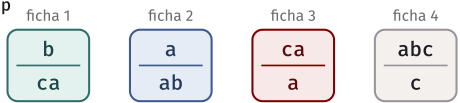




Ficha 2: arriba a, abajo ab. Abajo va adelantado: sobra b.

La solución, ficha por ficha

Las fichas de `sipser.pcp`



La fila

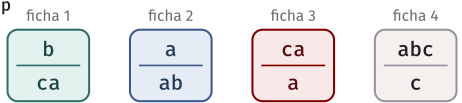




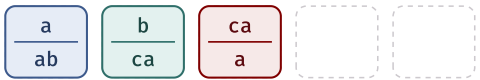
Ficha 1: arriba cubre la b que sobraba y abajo agrega ca. Ahora sobra ca.

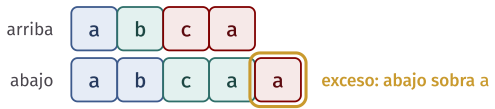
La solución, ficha por ficha

Las fichas de `sipser.pcp`



La fila

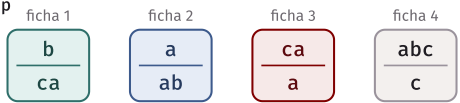




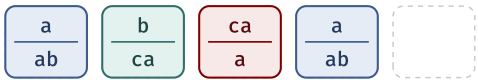
Ficha 3: arriba cubre `ca` y abajo agrega `a`. Sobra `a`.

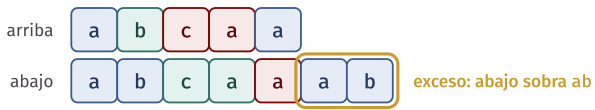
La solución, ficha por ficha

Las fichas de `sipser.pcp`



La fila

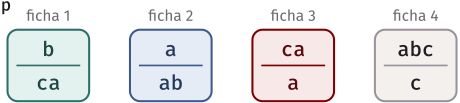




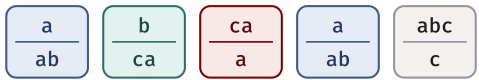
Ficha 2 otra vez: arriba cubre la a y abajo agrega ab. Sobra ab.

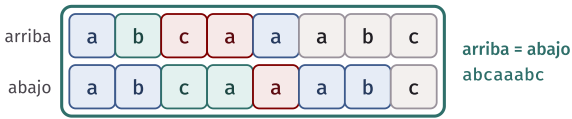
La solución, ficha por ficha

Las fichas de `sipser.pcp`



La fila

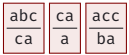




Ficha 4: abajo agrega `c`, y arriba pone justo `abc`. No sobra nada.

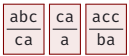
La secuencia 2 1 3 2 4 es una solución: 5 fichas, 8 símbolos. La ficha 2 se usa dos veces.

Tres instancias sin solución

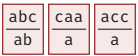


Ninguna puede ir primera: en las tres, arriba y abajo empiezan con símbolos distintos.

Tres instancias sin solución

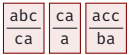


Ninguna puede ir primera: en las tres, arriba y abajo empiezan con símbolos distintos.

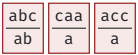


Arriba siempre es más largo: cualquier fila tiene arriba más símbolos que abajo.

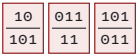
Tres instancias sin solución



Ninguna puede ir primera: en las tres, arriba y abajo empiezan con símbolos distintos.

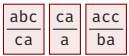


Arriba siempre es más largo: cualquier fila tiene arriba más símbolos que abajo.

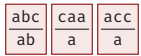


Se sigue para siempre: la ficha 1 deja abajo el exceso 1; solo cabe la ficha 3, que lo deja igual.

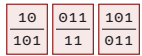
Tres instancias sin solución



Ninguna puede ir primera: en las tres, arriba y abajo empiezan con símbolos distintos.



Arriba siempre es más largo: cualquier fila tiene arriba más símbolos que abajo.



Se sigue para siempre: la ficha 1 deja abajo el exceso 1; solo cabe la ficha 3, que lo deja igual.

Tres instancias, tres argumentos distintos

Es lo esperable en un problema indecidible: no hay un argumento que sirva para todas.

PCP es reconocible

B = "Con la entrada $\langle P \rangle$, de k fichas:

1. Para $n = 1, 2, 3, \ldots$:
2. para cada secuencia de n fichas:
3. si arriba y abajo se lee lo mismo, *aceptar*."

PCP es reconocible

B = "Con la entrada $\langle P \rangle$, de k fichas:

1. Para $n = 1, 2, 3, \ldots$:
2. para cada secuencia de n fichas:
3. si arriba y abajo se lee lo mismo, *aceptar*."

- Si hay solución de n fichas, B llega a la vuelta n y acepta.
- Si no hay, B no se detiene. Reconoce, no decide.

PCP es reconocible

B = "Con la entrada $\langle P \rangle$, de k fichas:

1. Para $n = 1, 2, 3, \ldots$:
2. para cada secuencia de n fichas:
3. si arriba y abajo se lee lo mismo, *aceptar*."

- Si hay solución de n fichas, B llega a la vuelta n y acepta.
- Si no hay, B no se detiene. Reconoce, no decide.

Buscar mejor: un grafo de excesos

- Vértices: (lado adelantado, exceso). Aristas: las fichas.
- Búsqueda en anchura, por niveles.
- **No repetir excesos:** lo que sigue depende solo del exceso.
- **Una cota N:** el grafo puede ser infinito.

PCP es reconocible

B = "Con la entrada $\langle P \rangle$, de k fichas:

1. Para $n = 1, 2, 3, \ldots$:
2. para cada secuencia de n fichas:
3. si arriba y abajo se lee lo mismo, *aceptar*."

- Si hay solución de n fichas, B llega a la vuelta n y acepta.
- Si no hay, B no se detiene. Reconoce, no decide.

Buscar mejor: un grafo de excesos

- Vértices: (lado adelantado, exceso). Aristas: las fichas.
- Búsqueda en anchura, por niveles.
- **No repetir excesos:** lo que sigue depende solo del exceso.
- **Una cota N:** el grafo puede ser infinito.

El laboratorio

Es esta búsqueda. Responde con una solución o con `sin solución hasta N`.

Tres finales distintos

terminal

```
$ build/pcp_solve instances/sipser.pcp 4
4 fichas, hasta 4 por solución
sin solución hasta 4
excesos visitados: 4
$ build/pcp_solve instances/cycle.pcp 1000
3 fichas, hasta 1000 por solución
sin solución: la búsqueda se quedó sin excesos nuevos
excesos visitados: 1
$ build/pcp_solve instances/hard_75.pcp 400
3 fichas, hasta 400 por solución
búsqueda abandonada: demasiados excesos distintos
excesos visitados: 2000000
```

- `sipser`: tiene solución, de 5 fichas. Con cota 4, el programa solo sabe “hasta 4”.

Tres finales distintos

terminal

```
$ build/pcp_solve instances/sipser.pcp 4
4 fichas, hasta 4 por solución
sin solución hasta 4
excesos visitados: 4
$ build/pcp_solve instances/cycle.pcp 1000
3 fichas, hasta 1000 por solución
sin solución: la búsqueda se quedó sin excesos nuevos
excesos visitados: 1
$ build/pcp_solve instances/hard_75.pcp 400
3 fichas, hasta 400 por solución
búsqueda abandonada: demasiados excesos distintos
excesos visitados: 2000000
```

- `sipser`: tiene solución, de 5 fichas. Con cota 4, el programa solo sabe "hasta 4".
- `cycle`: el grafo alcanzable es **finito**. La búsqueda lo recorre y puede decir que no.

Tres finales distintos

terminal

```
$ build/pcp_solve instances/sipser.pcp 4
4 fichas, hasta 4 por solución
sin solución hasta 4
excesos visitados: 4
$ build/pcp_solve instances/cycle.pcp 1000
3 fichas, hasta 1000 por solución
sin solución: la búsqueda se quedó sin excesos nuevos
excesos visitados: 1
$ build/pcp_solve instances/hard_75.pcp 400
3 fichas, hasta 400 por solución
búsqueda abandonada: demasiados excesos distintos
excesos visitados: 2000000
```

- `sipser`: tiene solución, de 5 fichas. Con cota 4, el programa solo sabe "hasta 4".
- `cycle`: el grafo alcanzable es **finito**. La búsqueda lo recorre y puede decir que no.
- `hard_75`: $\left[\frac{110}{1}\right]\left[\frac{1}{0}\right]\left[\frac{0}{110}\right]$ los excesos se multiplican y el programa abandona.

Tres finales distintos

terminal

```
$ build/pcp_solve instances/sipser.pcp 4
4 fichas, hasta 4 por solución
sin solución hasta 4
excesos visitados: 4
$ build/pcp_solve instances/cycle.pcp 1000
3 fichas, hasta 1000 por solución
sin solución: la búsqueda se quedó sin excesos nuevos
excesos visitados: 1
$ build/pcp_solve instances/hard_75.pcp 400
3 fichas, hasta 400 por solución
búsqueda abandonada: demasiados excesos distintos
excesos visitados: 2000000
```

- `sipser`: tiene solución, de 5 fichas. Con cota 4, el programa solo sabe "hasta 4".
- `cycle`: el grafo alcanzable es **finito**. La búsqueda lo recorre y puede decir que no.
- `hard_75`: $\frac{110}{1}\ \frac{1}{0}\ \frac{0}{110}$ los excesos se multiplican y el programa abandona.
- Tiene solución: la más corta usa **75 fichas**. La encuentra `pcp_meet`, buscando desde los dos extremos.

“Todas las que probé tenían respuesta”

- Cada instancia de PCP **tiene** respuesta: o existe una solución o no existe.

“Todas las que probé tenían respuesta”

- Cada instancia de PCP **tiene** respuesta: o existe una solución o no existe.
- Para muchas instancias la respuesta se encuentra: con una búsqueda, o con un argumento a medida.

“Todas las que probé tenían respuesta”

- Cada instancia de PCP **tiene** respuesta: o existe una solución o no existe.
- Para muchas instancias la respuesta se encuentra: con una búsqueda, o con un argumento a medida.
- **Decidible** quiere decir otra cosa: que *un solo* algoritmo responda bien en *todas*.

“Todas las que probé tenían respuesta”

- Cada instancia de PCP **tiene** respuesta: o existe una solución o no existe.
- Para muchas instancias la respuesta se encuentra: con una búsqueda, o con un argumento a medida.
- **Decidible** quiere decir otra cosa: que *un solo* algoritmo responda bien en *todas*.

Error común

Creer que un problema es decidible porque todas las instancias que uno probó tenían respuesta. Un programa mejor mueve la frontera. No la elimina.

SECCIÓN 09

PCP es indecidible

09

PCP es indecidible: el plan

Reducción desde A_{TM} hacia PCP. De $\langle M, w \rangle$ se construyen fichas P tales que

$$M \text{ acepta } w \iff P \text{ tiene una solución.}$$

PCP es indecidible: el plan

Reducción desde A_{TM} hacia PCP. De $\langle M, w \rangle$ se construyen fichas P tales que

$$M \text{ acepta } w \iff P \text{ tiene una solución.}$$

- La única fila que coincide escribe una **historia de aceptación**, arriba y abajo.
- Abajo va siempre **una configuración adelantado**: al cubrir C_i arriba, las fichas escriben C_{i+1} abajo.

PCP es indecidible: el plan

Reducción desde A_{TM} hacia PCP. De $\langle M, w \rangle$ se construyen fichas P tales que

$$M \text{ acepta } w \iff P \text{ tiene una solución.}$$

- La única fila que coincide escribe una **historia de aceptación**, arriba y abajo.
- Abajo va siempre **una configuración adelantado**: al cubrir C_i arriba, las fichas escriben C_{i+1} abajo.
- Las fichas hacen, con cadenas, la revisión por ventanas.

PCP es indecidible: el plan

Reducción desde A_{TM} hacia PCP. De $\langle M, w \rangle$ se construyen fichas P tales que

$$M \text{ acepta } w \iff P \text{ tiene una solución.}$$

- La única fila que coincide escribe una **historia de aceptación**, arriba y abajo.
- Abajo va siempre **una configuración adelantado**: al cubrir C_i arriba, las fichas escriben C_{i+1} abajo.
- Las fichas hacen, con cadenas, la revisión por ventanas.

Dos simplificaciones:

1. **PCP modificado:** la solución tiene que empezar con la ficha 1. Se quita al final.
2. M nunca intenta moverse a la izquierda desde la primera celda. (Toda máquina se puede cambiar por una equivalente que lo cumple, con una marca en la primera celda.)

Las fichas de un ejemplo

terminal

```
$ build/tm_to_pcp machines/starts_01.tm 01
estados: A = q0, B = q1, C = q2, D = q_accept, E = q_reject
 n.º  tipo        arriba  abajo
   1  inicio      #       #A01#
   2  derecha     A0      0B
   3  derecha     C0      0D
   4  izquierda   0B1     C01
   5  izquierda   1B1     C11
   6  izquierda   _B1     C_1
   7  copia       0       0
   8  copia       1       1
   9  copia       _       _
  10  separador   #       #
  11  separador   #       _#
  12  comer       0D      D
  13  comer       1D      D
  14  comer       _D      D
  15  comer       D0      D
  16  comer       D1      D
  17  comer       D_      D
  18  final       D##     #
```

M acepta las cadenas que empiezan con `01`:

$$q_0 01 \;\to\; 0 q_1 1 \;\to\; q_2 01 \;\to\; 0 q_{\text{accept}} 1$$

- En una ficha cada símbolo es un carácter: cada estado recibe una letra.
- $|\Gamma| = 3$:
  $1 + 2 + 1 \cdot 3 + 3 + 2 + 2 \cdot 3 + 1 = 18$.
- No hay fichas para q_{reject} ni para un par (q, a) sin transición.

Los siete tipos

Tipo	Ficha	Para qué
1. Inicio	$\left[\begin{array}{c}\#\\ \hline \# \, q_0 \, w_1 \cdots w_n \, \#\end{array}\right]$	Deja abajo C_1. Desde aquí abajo va una configuración adelantado.

Los siete tipos

Tipo	Ficha	Para qué
1. Inicio	$\left[\dfrac{\#}{\#\, q_0\, w_1 \cdots w_n\, \#}\right]$	Deja abajo C_1. Desde aquí abajo va una configuración adelantado.
2. Derecha	$\left[\dfrac{q\,a}{b\,r}\right]$	Una por cada $\delta(q,a) = (r,b,\mathrm{R})$.
3. Izquierda	$\left[\dfrac{c\,q\,a}{r\,c\,b}\right]$	Por cada $\delta(q,a) = (r,b,\mathrm{L})$ y cada $c \in \Gamma$: el estado salta sobre c.

Los siete tipos

Tipo	Ficha	Para qué
1. Inicio	$\left[\frac{\#}{\# \, q_0 \, w_1 \cdots w_n \, \#}\right]$	Deja abajo C_1. Desde aquí abajo va una configuración adelantado.
2. Derecha	$\left[\frac{q \, a}{b \, r}\right]$	Una por cada $\delta(q, a) = (r, b, \mathrm{R})$.
3. Izquierda	$\left[\frac{c \, q \, a}{r \, c \, b}\right]$	Por cada $\delta(q, a) = (r, b, \mathrm{L})$ y cada $c \in \Gamma$: el estado salta sobre c.
4. Copia	$\left[\frac{a}{a}\right]$	Para cada $a \in \Gamma$: lo que está lejos del cabezal.
5. Separador	$\left[\frac{\#}{\#}\right], \left[\frac{\#}{\sqcup \, \#}\right]$	La segunda alarga la configuración con un blanco.

Los siete tipos

Tipo	Ficha	Para qué
1. Inicio	$\left[\frac{\#}{\# \, q_0 \, w_1 \cdots w_n \, \#}\right]$	Deja abajo C_1. Desde aquí abajo va una configuración adelantado.
2. Derecha	$\left[\frac{q \, a}{b \, r}\right]$	Una por cada $\delta(q, a) = (r, b, \mathrm{R})$.
3. Izquierda	$\left[\frac{c \, q \, a}{r \, c \, b}\right]$	Por cada $\delta(q, a) = (r, b, \mathrm{L})$ y cada $c \in \Gamma$: el estado salta sobre c.
4. Copia	$\left[\frac{a}{a}\right]$	Para cada $a \in \Gamma$: lo que está lejos del cabezal.
5. Separador	$\left[\frac{\#}{\#}\right], \left[\frac{\#}{\sqcup \, \#}\right]$	La segunda alarga la configuración con un blanco.
6. Comer	$\left[\frac{a \, q_{\text{accept}}}{q_{\text{accept}}}\right], \left[\frac{q_{\text{accept}} \, a}{q_{\text{accept}}}\right]$	Las primeras con arriba más largo: dejan que arriba alcance a abajo.
7. Final	$\left[\frac{q_{\text{accept}} \, \# \, \#}{\#}\right]$	Cierra cuando abajo solo queda $q_{\text{accept}} \, \#$.

La solución, vuelta por vuelta

inicio $\dfrac{\#}{\#A01\#}$

La solución, vuelta por vuelta

inicio $\dfrac{\#}{\#A\circ 1\#}$

vuelta 1 $\dfrac{A\circ}{\circ B}$ $\dfrac{1}{1}$ $\dfrac{\#}{\#}$ de C_1 sale C_2: paso a la derecha

En cada vuelta, lo que arriba cubre es lo que abajo se escribió en la vuelta anterior.

La solución, vuelta por vuelta

inicio 

vuelta 1 

de C_1 sale C_2: paso a la derecha

vuelta 2 

de C_2 sale C_3: paso a la izquierda

En cada vuelta, lo que arriba cubre es lo que abajo se escribió en la vuelta anterior.

La solución, vuelta por vuelta

inicio

#
#A○1#

vuelta 1

A○	1	#
○B	1	#

de C_1 sale C_2: paso a la derecha

vuelta 2

○B1	#
C○1	#

de C_2 sale C_3: paso a la izquierda

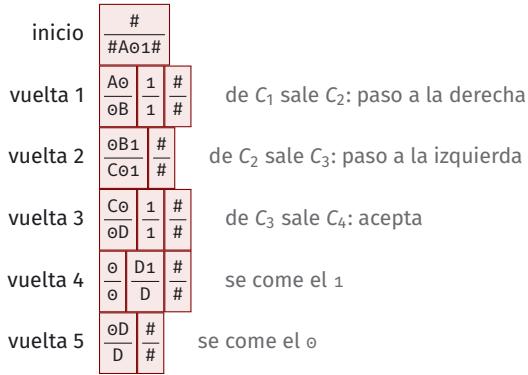
vuelta 3

C○	1	#
○D	1	#

de C_3 sale C_4: acepta

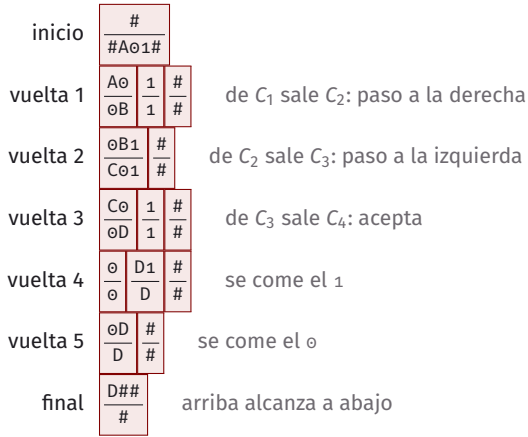
En cada vuelta, lo que arriba cubre es lo que abajo se escribió en la vuelta anterior.

La solución, vuelta por vuelta



En cada vuelta, lo que arriba cubre es lo que abajo se escribió en la vuelta anterior.

La solución, vuelta por vuelta



En cada vuelta, lo que arriba cubre es lo que abajo se escribió en la vuelta anterior.

Abajo va una configuración adelantado

inicio transición copia separador comer final $A = q_0$, $B = q_1$, $C = q_2$, $D = q_{accept}$

El número de cada pieza es el de su ficha: la misma ficha escribe arriba y abajo, en columnas distintas.

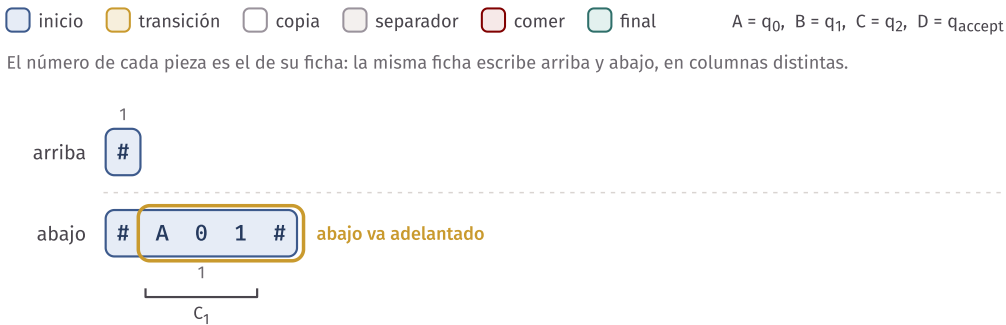
arriba

abajo

Para que arriba y abajo coincidan, la fila tiene que escribir la historia de `starts_01.tm` con `01`.

Historia: $q_0 01 \# 0 q_1 1 \# q_2 01 \# 0 q_{accept} 1$. En las fichas cada estado es una letra.

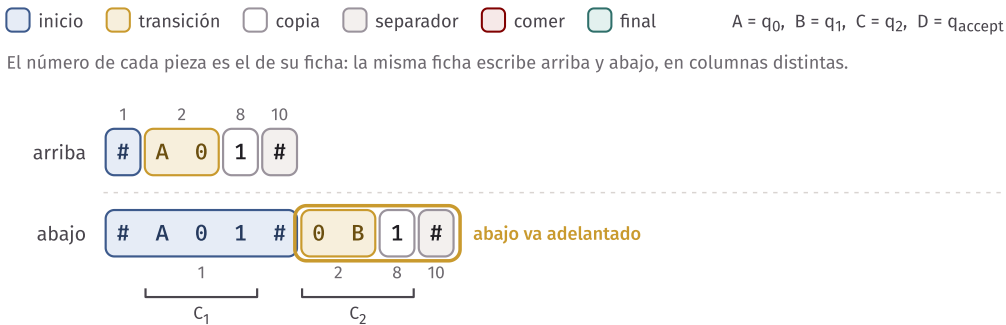
Abajo va una configuración adelantado



Ficha 1, la de inicio: arriba pone solo # y abajo deja escrita $C_1 = q_0 01$.

Desde aquí abajo va adelantado en una configuración entera.

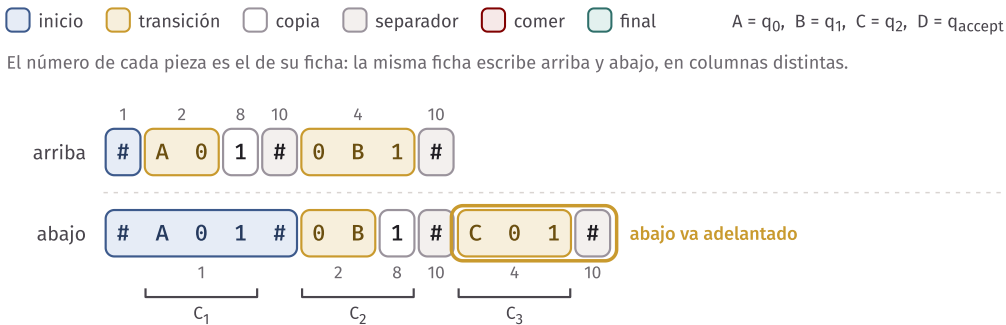
Abajo va una configuración adelantado



Vuelta 1, fichas 2, 8 y 10: arriba cubre C_1 y abajo escribe C_2.

La ficha 2 es la transición $\delta(q_0, 0) = (q_1, 0, R)$; la 8 copia el 1 y la 10, el separador.

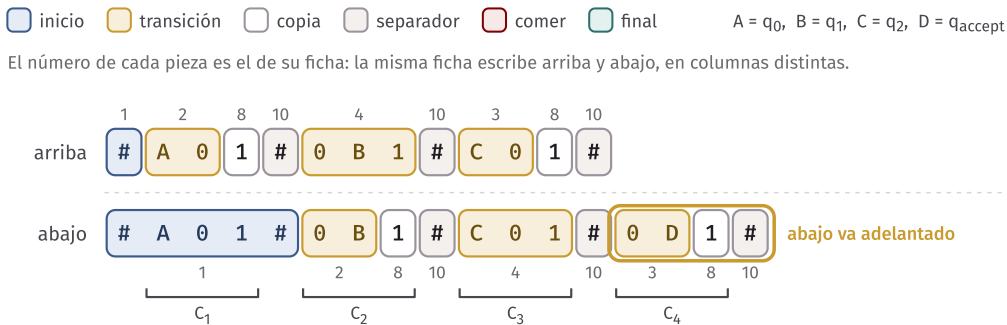
Abajo va una configuración adelantado



Vuelta 2, fichas 4 y 10: arriba cubre C_2 y abajo escribe C_3.

La ficha 4 es el paso a la izquierda $\delta(q_1, 1) = (q_2, 1, L)$: el estado salta sobre el `0`.

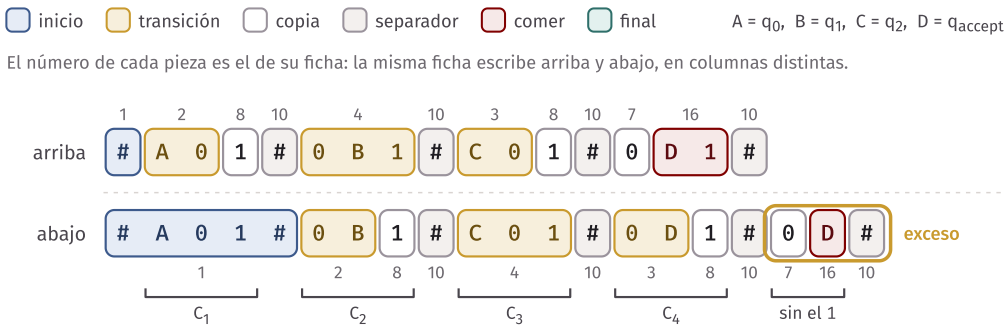
Abajo va una configuración adelantado



Vuelta 3, fichas 3, 8 y 10: arriba cubre C_3 y abajo escribe C_4, que tiene a q_{accept}.

La ficha 3 es $\delta(q_2, 0) = (q_{accept}, 0, R)$. La máquina aceptó, pero abajo sigue adelantado.

Abajo va una configuración adelantado



Vuelta 4, fichas 7, 16 y 10: ya no se simula nada. La ficha 16 se come el 1.

Abajo se escribe la misma configuración con un símbolo menos: el exceso se acorta.

Abajo va una configuración adelantado



Vuelta 5, fichas 12 y 10: la ficha 12 se come el `0`.

Abajo solo queda por cubrir D#, el estado de aceptación y un separador.

Abajo va una configuración adelantado



Ficha 18, la final: arriba cubre D# y los dos separadores del final se emparejan.

Arriba alcanzó a abajo: las 15 fichas son una solución, y la cadena empieza con la historia de cómputo.

La búsqueda la encuentra

terminal

```
$ build/tm_to_pcp machines/starts_01.tm 01 --mpcp > build/starts_01.pcp
$ build/pcp_solve build/starts_01.pcp 30 --primera
18 fichas, hasta 30 por solución, empezando con la ficha 1
solución con 15 fichas: 1 2 8 10 4 10 3 8 10 7 16 10 12 10 18
cadena: #A01#0B1#C01#0D1#0D#D##
excesos visitados: 459
$ build/pcp_check build/starts_01.pcp 1 2 8 10 4 10 3 8 10 7 16 10 12 10 18
arriba: # A0 1 # 0B1 # C0 1 # 0 D1 # 0D # D##  =  #A01#0B1#C01#0D1#0D#D##
abajo:  #A01# 0B 1 # C01 # 0D 1 # 0 D # D # #  =  #A01#0B1#C01#0D1#0D#D##
es una solución (15 fichas, 23 símbolos)
```

- La cadena empieza con la historia: $\#\, q_0 01 \,\#\, 0 q_1 1 \,\#\, q_2 01 \,\#\, 0 q_{\text{accept}} 1 \,\#$. Lo demás es la limpieza.

Por qué funciona

Si M acepta w, hay solución. Al empezar la vuelta i:

$$\text{arriba: } \#C_1\#\cdots\#C_{i-1}\# \qquad \text{abajo: } \#C_1\#\cdots\#C_{i-1}\#C_i\#$$

Se cubre $C_i\#$ con la ficha de la transición, copias y un separador: abajo queda $C_{i+1}\#$. Al llegar a q_{accept}, comer y cerrar.

Por qué funciona

Si M acepta w, hay solución. Al empezar la vuelta i:

$$\text{arriba: } \#C_1\# \cdots \#C_{i-1}\# \qquad \text{abajo: } \#C_1\# \cdots \#C_{i-1}\#C_i\#$$

Se cubre $C_i\#$ con la ficha de la transición, copias y un separador: abajo queda $C_{i+1}\#$. Al llegar a q_{accept}, comer y cerrar.

Si hay solución, M acepta w.

- Con los tipos 1 a 5, arriba nunca es más largo que abajo: la solución usa una ficha con q_{accept} arriba.

Por qué funciona

Si M acepta w, hay solución. Al empezar la vuelta i:

$$\text{arriba: } \#C_1\#\cdots\#C_{i-1}\# \qquad \text{abajo: } \#C_1\#\cdots\#C_{i-1}\#C_i\#$$

Se cubre $C_i\#$ con la ficha de la transición, copias y un separador: abajo queda $C_{i+1}\#$. Al llegar a q_{accept}, comer y cerrar.

Si hay solución, M acepta w.

- Con los tipos 1 a 5, arriba nunca es más largo que abajo: la solución usa una ficha con q_{accept} arriba.
- Antes de eso las fichas no dejan elegir: un símbolo lejos del estado solo se cubre con su copia, y un estado q que lee a, solo con la ficha de $\delta(q, a)$. Lo que se escribe abajo es lo que M produce.

Por qué funciona

Si M acepta w, hay solución. Al empezar la vuelta i:

$$\text{arriba: } \#C_1\#\cdots\#C_{i-1}\# \qquad \text{abajo: } \#C_1\#\cdots\#C_{i-1}\#C_i\#$$

Se cubre $C_i\#$ con la ficha de la transición, copias y un separador: abajo queda $C_{i+1}\#$. Al llegar a q_{accept}, comer y cerrar.

Si hay solución, M acepta w.

- Con los tipos 1 a 5, arriba nunca es más largo que abajo: la solución usa una ficha con q_{accept} arriba.
- Antes de eso las fichas no dejan elegir: un símbolo lejos del estado solo se cubre con su copia, y un estado q que lee a, solo con la ficha de $\delta(q, a)$. Lo que se escribe abajo es lo que M produce.
- Entonces la primera configuración con q_{accept} es una que M alcanza desde C_1.

`test_tm_pcp`: 27 pares de máquina y entrada; hay solución en los 9 en que la máquina acepta, y ninguna de hasta 40 fichas en los otros 18.

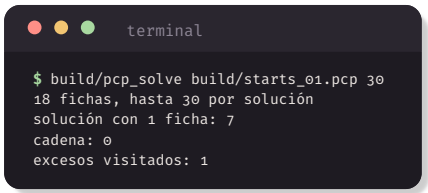
Cuando *M* no acepta

terminal

```
$ build/tm_to_pcp machines/starts_01.tm 00 --mpcp > build/starts_00.pcp
$ build/pcp_solve build/starts_00.pcp 30 --primera
18 fichas, hasta 30 por solución, empezando con la ficha 1
sin solución: la búsqueda se quedó sin excesos nuevos
excesos visitados: 9
$ build/tm_to_pcp machines/loop_on_0.tm 00 --mpcp > build/loop_00.pcp
$ build/pcp_solve build/loop_00.pcp 60 --primera
17 fichas, hasta 60 por solución, empezando con la ficha 1
sin solución hasta 60
excesos visitados: 20359
```

- `starts_01` **rechaza** `00`: la fila se atasca donde se atasca la máquina.
- `loop_on_0` **no se detiene** con `00`: las fichas tampoco, con ninguna cota.
- Buscar una solución de estas fichas *es* simular a *M* con *w*, con otro formato.

Quitar la condición de la primera ficha



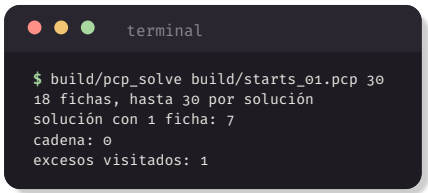
Con dos símbolos nuevos, $*$ y $\diamond$, las fichas pasan a ser

$$\left[\frac{*t_1}{*b_1*}\right], \quad \left[\frac{*t_i}{b_i*}\right] \text{ para todo } i, \quad \left[\frac{*\diamond}{\diamond}\right]$$

Sin la condición, una copia ya es solución.

- $*u$: un asterisco antes de cada símbolo de u. $u*$: uno después de cada símbolo.

Quitar la condición de la primera ficha



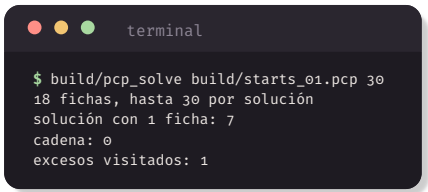
Sin la condición, una copia ya es solución.

Con dos símbolos nuevos, $*$ y $\diamond$, las fichas pasan a ser

$$\left[\frac{*t_1}{*b_1*}\right], \quad \left[\frac{*t_i}{b_i*}\right] \text{ para todo } i, \quad \left[\frac{*\diamond}{\diamond}\right]$$

- $*u$: un asterisco antes de cada símbolo de u. $u*$: uno después de cada símbolo.
- **Solo la primera puede empezar:** en las demás, arriba empieza con $*$ y abajo no.

Quitar la condición de la primera ficha



Sin la condición, una copia ya es solución.

Con dos símbolos nuevos, $*$ y $\diamond$, las fichas pasan a ser

$$\left[\frac{*t_1}{*b_1*}\right], \quad \left[\frac{*t_i}{b_i*}\right] \text{ para todo } i, \quad \left[\frac{*\diamond}{\diamond}\right]$$

- $*u$: un asterisco antes de cada símbolo de u. $u*$: uno después de cada símbolo.
- **Solo la primera puede empezar:** en las demás, arriba empieza con $*$ y abajo no.
- Después los asteriscos coinciden, y la última ficha pone el que falta arriba.

PCP es indecidible

- De $\langle M, w \rangle$ a las fichas con la primera fija, y de ahí a las fichas sin condición.

PCP es indecidible

- De $\langle M, w \rangle$ a las fichas con la primera fija, y de ahí a las fichas sin condición.
- En el ejemplo: 20 fichas, y la misma solución con una ficha más (16).

PCP es indecidible

- De $\langle M, w \rangle$ a las fichas con la primera fija, y de ahí a las fichas sin condición.
- En el ejemplo: 20 fichas, y la misma solución con una ficha más (16).

Teorema

$A_{\mathrm{TM}} \leq_m \mathrm{PCP}$. Entonces PCP es indecidible. Como es reconocible, su complemento no es reconocible.

PCP es indecidible

- De $\langle M, w \rangle$ a las fichas con la primera fija, y de ahí a las fichas sin condición.
- En el ejemplo: 20 fichas, y la misma solución con una ficha más (16).

Teorema

$A_{\mathrm{TM}} \leq_m \mathrm{PCP}$. Entonces PCP es indecidible. Como es reconocible, su complemento no es reconocible.

Lo que hay que llevarse

Las fichas no calculan: cada una solo dice "si arriba va esto, abajo va esto otro". Como comprobar un paso de una máquina es local, esas reglas alcanzan para obligar a que la fila sea una ejecución correcta.

SECCIÓN 10

Problemas indecidibles sobre gramáticas

10



$\mathrm{ALL}_{\mathrm{CFG}}$: ¿una gramática genera todas las cadenas?

El problema

$\mathrm{ALL}_{\mathrm{CFG}} = \{ \langle G \rangle : G$ es una gramática libre de contexto y $L(G) = \Sigma^* \}$

- Con un DFA: $L = \Sigma^*$ si y solo si el complemento es vacío. Con gramáticas no hay complemento.

$\mathrm{ALL}_{\mathrm{CFG}}$: ¿una gramática genera todas las cadenas?

El problema

$\mathrm{ALL}_{\mathrm{CFG}} = \{ \langle G \rangle : G \text{ es una gramática libre de contexto y } L(G) = \Sigma^* \}$

- Con un DFA: $L = \Sigma^*$ si y solo si el complemento es vacío. Con gramáticas no hay complemento.
- **Reducción desde A_{TM}.** De $\langle M, w \rangle$ se construye G que genera todo lo que no es una historia de aceptación de M con w.

$\mathrm{ALL}_{\mathrm{CFG}}$: ¿una gramática genera todas las cadenas?

El problema

$\mathrm{ALL}_{\mathrm{CFG}} = \{ \langle G \rangle : G \text{ es una gramática libre de contexto y } L(G) = \Sigma^* \}$

- Con un DFA: $L = \Sigma^*$ si y solo si el complemento es vacío. Con gramáticas no hay complemento.
- **Reducción desde A_{TM}.** De $\langle M, w \rangle$ se construye G que genera todo lo que no es una historia de aceptación de M con w.

$$M \text{ acepta } w \iff \text{existe la historia} \iff L(G) \neq \Sigma^*$$

$\mathrm{ALL}_{\mathrm{CFG}}$: ¿una gramática genera todas las cadenas?

El problema

$\mathrm{ALL}_{\mathrm{CFG}} = \{ \langle G \rangle : G \text{ es una gramática libre de contexto y } L(G) = \Sigma^* \}$

- Con un DFA: $L = \Sigma^*$ si y solo si el complemento es vacío. Con gramáticas no hay complemento.
- **Reducción desde A_{TM}.** De $\langle M, w \rangle$ se construye G que genera todo lo que no es una historia de aceptación de M con w.

$$M \text{ acepta } w \iff \text{existe la historia} \iff L(G) \neq \Sigma^*$$

- Ser una historia es una conjunción: *todos* los pasos encajan. Una pila no alcanza.
- **No** serlo es una disyunción: basta *un* error. Eso sí está al alcance de una pila.

$\mathrm{ALL}_{\mathrm{CFG}}$: ¿una gramática genera todas las cadenas?

El problema

$\mathrm{ALL}_{\mathrm{CFG}} = \{ \langle G \rangle : G \text{ es una gramática libre de contexto y } L(G) = \Sigma^* \}$

- Con un DFA: $L = \Sigma^*$ si y solo si el complemento es vacío. Con gramáticas no hay complemento.
- **Reducción desde A_{TM}.** De $\langle M, w \rangle$ se construye G que genera todo lo que no es una historia de aceptación de M con w.

$$M \text{ acepta } w \iff \text{existe la historia} \iff L(G) \neq \Sigma^*$$

- Ser una historia es una conjunción: *todos* los pasos encajan. Una pila no alcanza.
- **No** serlo es una disyunción: basta *un* error. Eso sí está al alcance de una pila.

Como función: $A_{\mathrm{TM}} \leq_m \overline{\mathrm{ALL}_{\mathrm{CFG}}}$. Entonces $\mathrm{ALL}_{\mathrm{CFG}}$ no es ni siquiera reconocible.

El autómata de pila que señala un error

D adivina cuál condición falla:

1. la forma no es de configuraciones con #;
2. la primera no es $q_0\, w$;
3. la última no tiene a q_{accept};
4. algún C_i no produce C_{i+1}.

1, 2 y 3: sin pila. 4: mete C_i en la pila y la saca leyendo C_{i+1}.

El truco: la pila devuelve C_i al revés. Se escriben al revés las configuraciones pares:

$$\# C_1 \# C_2^R \# C_3 \# C_4^R \# \cdots$$

Es por lo que una pila reconoce $\{w\#w^R\}$.

```
build/history machines/zero_n_one_n.tm 01 --alterna (última línea)

historia: q0 0 1 # 1 q1 X # q2 X Y # Y q0 X # X Y q3 _ # _ q_accept _ Y X
```

De *D* se obtiene *G* con el algoritmo de la semana 6. Todo es escribir descripciones: es computable.

Consecuencia: $\mathrm{EQ}_{\mathrm{CFG}}$ es indecidible

La deuda de la semana 9. **Reducción desde $\mathrm{ALL}_{\mathrm{CFG}}$ hacia $\mathrm{EQ}_{\mathrm{CFG}}$.**

- G_{todo}: una variable S, con $S \to aS$ para cada $a \in \Sigma$, y $S \to \varepsilon$. Genera Σ^*.

Consecuencia: $\mathrm{EQ}_{\mathrm{CFG}}$ es indecidible

La deuda de la semana 9. **Reducción desde $\mathrm{ALL}_{\mathrm{CFG}}$ hacia $\mathrm{EQ}_{\mathrm{CFG}}$.**

- G_{todo}: una variable S, con $S \to aS$ para cada $a \in \Sigma$, y $S \to \varepsilon$. Genera Σ^*.
- La función: $f(\langle G \rangle) = \langle G, G_{\text{todo}} \rangle$.

Consecuencia: $\mathrm{EQ}_{\mathrm{CFG}}$ es indecidible

La deuda de la semana 9. **Reducción desde $\mathrm{ALL}_{\mathrm{CFG}}$ hacia $\mathrm{EQ}_{\mathrm{CFG}}$.**

- G_{todo}: una variable S, con $S \to aS$ para cada $a \in \Sigma$, y $S \to \varepsilon$. Genera Σ^*.
- La función: $f(\langle G \rangle) = \langle G, G_{\text{todo}} \rangle$.
- $\langle G \rangle \in \mathrm{ALL}_{\mathrm{CFG}} \iff L(G) = \Sigma^* = L(G_{\text{todo}}) \iff \langle G, G_{\text{todo}} \rangle \in \mathrm{EQ}_{\mathrm{CFG}}$.

Consecuencia: $\mathrm{EQ}_{\mathrm{CFG}}$ es indecidible

La deuda de la semana 9. **Reducción desde $\mathrm{ALL}_{\mathrm{CFG}}$ hacia $\mathrm{EQ}_{\mathrm{CFG}}$.**

- G_{todo}: una variable S, con $S \to aS$ para cada $a \in \Sigma$, y $S \to \varepsilon$. Genera Σ^*.
- La función: $f(\langle G \rangle) = \langle G, G_{\text{todo}} \rangle$.
- $\langle G \rangle \in \mathrm{ALL}_{\mathrm{CFG}} \iff L(G) = \Sigma^* = L(G_{\text{todo}}) \iff \langle G, G_{\text{todo}} \rangle \in \mathrm{EQ}_{\mathrm{CFG}}$.
- Es el mismo movimiento que llevó de E_{TM} a $\mathrm{EQ}_{\mathrm{TM}}$.

Consecuencia: $\mathrm{EQ}_{\mathrm{CFG}}$ es indecidible

La deuda de la semana 9. **Reducción desde $\mathrm{ALL}_{\mathrm{CFG}}$ hacia $\mathrm{EQ}_{\mathrm{CFG}}$.**

- G_{todo}: una variable S, con $S \to aS$ para cada $a \in \Sigma$, y $S \to \varepsilon$. Genera Σ^*.
- La función: $f(\langle G \rangle) = \langle G, G_{\text{todo}} \rangle$.
- $\langle G \rangle \in \mathrm{ALL}_{\mathrm{CFG}} \iff L(G) = \Sigma^* = L(G_{\text{todo}}) \iff \langle G, G_{\text{todo}} \rangle \in \mathrm{EQ}_{\mathrm{CFG}}$.
- Es el mismo movimiento que llevó de E_{TM} a $\mathrm{EQ}_{\mathrm{TM}}$.

En la práctica

Dos gramáticas del mismo lenguaje de programación no se pueden comparar con un algoritmo general. Si difieren, probando cadenas se encuentra una que lo muestra. Lo que no se obtiene es la confirmación de que son iguales.

Consecuencia: decidir si una gramática es ambigua

Indecidible. Se enuncia, con la idea: una **reducción desde PCP**.

$$S \to T \mid B$$
$$T \to t_i\, T\, \mathrm{d}_i \mid t_i\, \mathrm{d}_i$$
$$B \to b_i\, B\, \mathrm{d}_i \mid b_i\, \mathrm{d}_i$$

para $i = 1, \ldots, k$; d_i es una etiqueta nueva por ficha.

Consecuencia: decidir si una gramática es ambigua

Indecidible. Se enuncia, con la idea: una **reducción desde PCP**.

$$S \to T \mid B$$
$$T \to t_i \, T \, \mathrm{d}_i \mid t_i \, \mathrm{d}_i$$
$$B \to b_i \, B \, \mathrm{d}_i \mid b_i \, \mathrm{d}_i$$

para $i = 1, \ldots, k$; d_i es una etiqueta nueva por ficha.

- Desde T: lo de arriba de una fila, y las etiquetas de sus fichas al revés. Desde B: lo de abajo.

Consecuencia: decidir si una gramática es ambigua

Indecidible. Se enuncia, con la idea: una **reducción desde PCP**.

$$S \to T \mid B$$
$$T \to t_i \, T \, \mathrm{d}_i \mid t_i \, \mathrm{d}_i$$
$$B \to b_i \, B \, \mathrm{d}_i \mid b_i \, \mathrm{d}_i$$

para $i = 1, \ldots, k$; d_i es una etiqueta nueva por ficha.

- Desde T: lo de arriba de una fila, y las etiquetas de sus fichas al revés. Desde B: lo de abajo.
- Las etiquetas dicen qué reglas se usaron: desde T sola, o desde B sola, un solo árbol.

Consecuencia: decidir si una gramática es ambigua

Indecidible. Se enuncia, con la idea: una **reducción desde PCP**.

$$S \to T \mid B$$
$$T \to t_i\, T\, \mathrm{d}_i \mid t_i\, \mathrm{d}_i$$
$$B \to b_i\, B\, \mathrm{d}_i \mid b_i\, \mathrm{d}_i$$

para $i = 1, \ldots, k$; d_i es una etiqueta nueva por ficha.

- Desde T: lo de arriba de una fila, y las etiquetas de sus fichas al revés. Desde B: lo de abajo.
- Las etiquetas dicen qué reglas se usaron: desde T sola, o desde B sola, un solo árbol.
- Dos árboles $\iff$ la misma fila y la misma cadena arriba y abajo $\iff$ una solución.

Consecuencia: decidir si una gramática es ambigua

Indecidible. Se enuncia, con la idea: una **reducción desde PCP**.

$$S \to T \mid B$$
$$T \to t_i\, T\, \mathrm{d}_i \mid t_i\, \mathrm{d}_i$$
$$B \to b_i\, B\, \mathrm{d}_i \mid b_i\, \mathrm{d}_i$$

para $i = 1, \ldots, k$; d_i es una etiqueta nueva por ficha.

- Desde T: lo de arriba de una fila, y las etiquetas de sus fichas al revés. Desde B: lo de abajo.
- Las etiquetas dicen qué reglas se usaron: desde T sola, o desde B sola, un solo árbol.
- Dos árboles $\iff$ la misma fila y la misma cadena arriba y abajo $\iff$ una solución.

Para qué sirve PCP

Reducir desde PCP hacia un problema de gramáticas es mucho más corto que hacerlo desde A_{TM}.

El contraste con los autómatas finitos

Pregunta	DFA	Gramática libre de contexto	Máquina de Turing
¿Acepta (genera) w?	decidible	decidible	indecidible (A_{TM})
¿El lenguaje es vacío?	decidible	decidible	indecidible (E_{TM})
¿El lenguaje es Σ^*?	decidible	indecidible ($\mathrm{ALL}_{\mathrm{CFG}}$)	indecidible
¿Dos son equivalentes?	decidible	indecidible ($\mathrm{EQ}_{\mathrm{CFG}}$)	indecidible ($\mathrm{EQ}_{\mathrm{TM}}$)
¿El lenguaje es regular?	siempre sí	indecidible (se enuncia)	indecidible ($\mathrm{REGULAR}_{\mathrm{TM}}$)

El contraste con los autómatas finitos

Pregunta	DFA	Gramática libre de contexto	Máquina de Turing
¿Acepta (genera) w?	decidible	decidible	indecidible (A_{TM})
¿El lenguaje es vacío?	decidible	decidible	indecidible (E_{TM})
¿El lenguaje es Σ^*?	decidible	indecidible ($\mathrm{ALL}_{\mathrm{CFG}}$)	indecidible
¿Dos son equivalentes?	decidible	indecidible ($\mathrm{EQ}_{\mathrm{CFG}}$)	indecidible ($\mathrm{EQ}_{\mathrm{TM}}$)
¿El lenguaje es regular?	siempre sí	indecidible (se enuncia)	indecidible ($\mathrm{REGULAR}_{\mathrm{TM}}$)

- Gramáticas: mirar *dentro* del lenguaje es decidible. Mirar lo que *no* generan, no.

El contraste con los autómatas finitos

Pregunta	DFA	Gramática libre de contexto	Máquina de Turing
¿Acepta (genera) w?	decidible	decidible	indecidible (A_{TM})
¿El lenguaje es vacío?	decidible	decidible	indecidible (E_{TM})
¿El lenguaje es Σ^*?	decidible	indecidible ($\mathrm{ALL}_{\mathrm{CFG}}$)	indecidible
¿Dos son equivalentes?	decidible	indecidible ($\mathrm{EQ}_{\mathrm{CFG}}$)	indecidible ($\mathrm{EQ}_{\mathrm{TM}}$)
¿El lenguaje es regular?	siempre sí	indecidible (se enuncia)	indecidible ($\mathrm{REGULAR}_{\mathrm{TM}}$)

- Gramáticas: mirar *dentro* del lenguaje es decidible. Mirar lo que *no* generan, no.
- Más poder de expresión, menos posibilidad de análisis. Quien diseña un formato de reglas elige su columna.

SECCIÓN 11

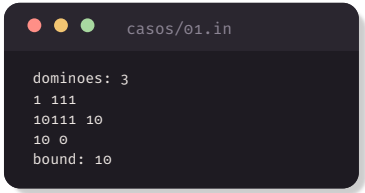
Laboratorio

11

Laboratorio: buscar una solución de PCP

1. Leer las fichas y buscar una solución de a lo sumo N fichas.
2. Buscar en anchura sobre el exceso.
3. No repetir excesos ya visitados.
4. Responder con la secuencia, o con `sin solución hasta N`.
5. Explicar por qué el programa reconoce PCP y no lo decide.

Dónde: `code/week-10/lab/pcp_search`. Varias respuestas son correctas: el juez usa un verificador. Probar todas las secuencias da TLE.



Una salida correcta: `2 1 1 3`

Para la próxima clase

- Leer los apuntes de la semana 10 y terminar el laboratorio.
- Preguntas de repaso de la guía: "M acepta ε" con una reducción y con Rice; cuáles propiedades caen bajo Rice; qué se sabe de A y de B si $A \leq_m B$; un PCP de cuatro fichas; por qué comprobar una historia es local.
- Próxima semana: autorreferencia. El teorema de recursión, los quines y la incompletitud de Gödel.

Pregunta

La máquina D de la semana 9 recibía su propia descripción. ¿Puede un programa obtener su propio código sin leer su archivo? Intenten escribir uno que lo imprima.