

Teoría de la Computación

Máquinas de Turing

Configuraciones, reconocer y decidir, variantes y la tesis de Church-Turing

Gerson Garrido

Semana 7

UNIVERSIDAD NACIONAL DE INGENIERÍA

Hoy

Sesión A

1. Lo que faltaba: escribir y volver
2. La definición formal
3. Configuraciones
4. Aceptar, rechazar o no detenerse
5. Tres máquinas, paso a paso
6. Reconocer y decidir
7. Los tres niveles de descripción

Sesión B

8. El modelo es robusto
9. Máquinas de varias cintas
10. Máquinas no deterministas
11. Enumeradores
12. Otros modelos equivalentes
13. La tesis de Church-Turing
14. Codificar objetos como cadenas

Lectura: apuntes de la semana 7. Apoyo: Sipser, sec. 3.1, 3.2 y 3.3; Erickson, *Models of Computation*, nota 6. Código en `code/week-07`.

SECCIÓN 01

Lo que les faltaba a los modelos anteriores

01

De dónde venimos

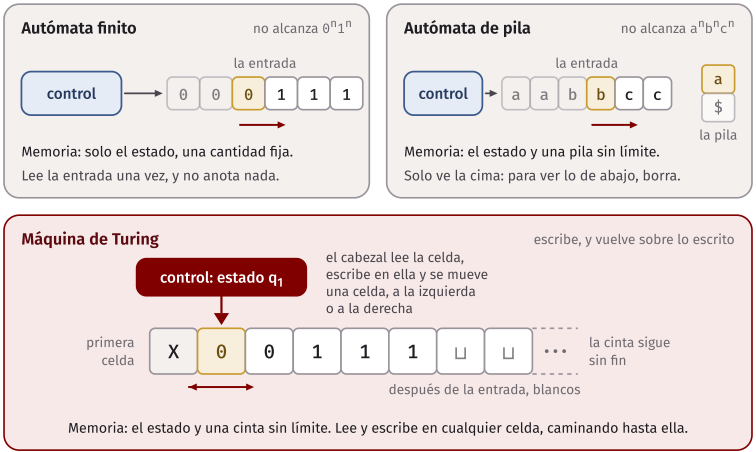
Modelo	Memoria	Cómo la usa	No alcanza
DFA, NFA	El estado: cantidad fija	Lee una vez, de izquierda a derecha	$\{0^n 1^n\}$
PDA	El estado y una pila	Solo ve la cima	$\{a^n b^n c^n\}$
Máquina de Turing	El estado y una cinta	Lee y escribe donde quiera, y vuelve	Semana 9

- Al autómata finito le falta dónde **anotar**.
- Al autómata de pila le falta **volver** sobre lo anotado sin destruirlo.
- Y cualquiera de ustedes decide $\{a^n b^n c^n\}$ con un programa de diez líneas.

Parte 3: computabilidad

Un modelo que no se queda corto frente a una computadora. Con él, “no existe ningún algoritmo” se puede demostrar.

Tres modelos, tres memorias



La máquina: cinta, cabezal y control

- **Cinta** *(tape)*: celdas con un símbolo cada una. Empieza en una primera celda y no termina.
- **Cabezal** *(head)*: lee la celda donde está, la cambia y se mueve *una* celda.
- **Control**: una cantidad finita de estados, como un DFA.
- Al empezar: la entrada en las primeras celdas, el resto en **blanco** (␣), el cabezal en la primera.
- En cada paso mira el estado y el símbolo, y hace tres cosas: escribe, se mueve, cambia de estado.

Cuatro diferencias con un autómata finito

1. La máquina **escribe** en la cinta, además de leerla.
2. El cabezal se mueve **en los dos sentidos**: la entrada se puede leer cuantas veces haga falta.
3. La cinta **no se acaba**: después de la entrada hay todos los blancos que la máquina quiera usar.
4. Aceptar y rechazar **tienen efecto de inmediato**: al entrar a uno de esos estados, la máquina se detiene.

La novedad incómoda

Un DFA responde al terminar de leer. Una máquina de Turing responde cuando quiere. O no responde.

SECCIÓN 02

La definición formal

02

La definición formal

Máquina de Turing *(Turing machine)*

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, con Q, Σ, Γ finitos:

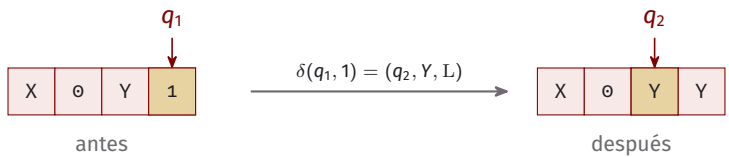
1. Q: los **estados**.
2. Σ: el **alfabeto de entrada**. No contiene a $\sqcup$.
3. Γ: el **alfabeto de la cinta**. $\sqcup \in \Gamma$ y $\Sigma \subseteq \Gamma$.
4. $\delta : Q' \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$, con $Q' = Q - \{q_{\text{accept}}, q_{\text{reject}}\}$.
5. q_0: el estado inicial. 6. q_{accept}: el de aceptación. 7. $q_{\text{reject}} \neq q_{\text{accept}}$: el de rechazo.

- Dos alfabetos: la cinta tiene además marcas y tachaduras. El primer blanco marca el final de la entrada.
- De q_{accept} y de q_{reject} no sale nada. Q y Γ son finitos: lo que no tiene tope es la cinta.

Leer una transición

$\delta(q, a) = (r, b, \mathrm{R})$

En el estado q, leyendo a: **escribe** b, **mueve** el cabezal una celda a la derecha y **pasa** a r.



- Las tres cosas pasan en *un* paso, y siempre pasan las tres.
- Para "no escribir", se escribe el mismo símbolo: $b = a$.
- δ es una función: una sola acción por cada estado y símbolo. La máquina es **determinista**.

Dos acuerdos del curso

El borde izquierdo

La cinta tiene una primera celda. Un movimiento L ahí **deja el cabezal donde está**.

- La máquina igual escribe y cambia de estado.
- No se entera: si necesita saber dónde empieza la cinta, deja una marca.

Las casillas vacías

Un par (q, a) sin transición escrita **lleva a** q_{reject}:
$\delta(q, a) = (q_{\mathrm{reject}}, a, \mathrm{R})$

- Así δ sigue siendo una función total.
- Es el estado trampa de la semana 1, sin dibujarlo.

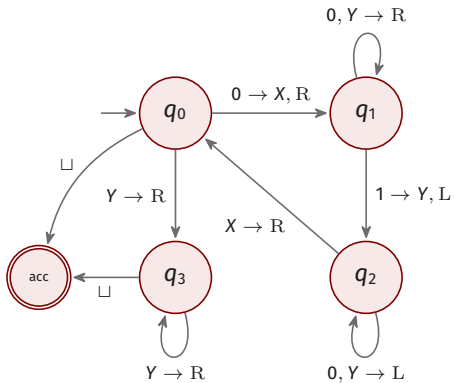
Como en Sipser y en `code/formats.md`. El laboratorio usa exactamente estos dos acuerdos.

La primera máquina: $\{0^n 1^n : n \geq 0\}$

La idea, con lápiz y papel:

1. Tachar el primer 0.
2. Buscar el primer 1 y tacharlo.
3. Volver, y repetir.
4. Si todo quedó tachado, aceptar.

q_0 primer símbolo sin tachar
q_1 busca un 1 hacia la derecha
q_2 vuelve hasta la última X
q_3 solo deben quedar Y



En el diagrama, $a \to b, \mathrm{R}$ es "lee a, escribe b, va a la derecha"; $a \to \mathrm{R}$ deja el símbolo. Lo que no está dibujado va a q_{reject}.

La tabla y el archivo dicen lo mismo

zeros_ones.tm

```
states: q0 q1 q2 q3 q_accept q_reject
alphabet: 0 1
tape: 0 1 X Y _
start: q0
accept: q_accept
reject: q_reject
delta:
q0 0 -> q1 X R
q0 Y -> q3 Y R
q0 _ -> q_accept _ R
q1 0 -> q1 0 R
q1 Y -> q1 Y R
q1 1 -> q2 Y L
q2 0 -> q2 0 L
q2 Y -> q2 Y L
q2 X -> q0 X R
q3 Y -> q3 Y R
q3 _ -> q_accept _ R
```

δ	0	1	X	Y	⊔
q_0	q_1, X, R			q_3, Y, R	$q_{\text{accept}}, ⊔, \mathrm{R}$
q_1	$q_1, 0, \mathrm{R}$	q_2, Y, L		q_1, Y, R	
q_2	$q_2, 0, \mathrm{L}$		q_0, X, R	q_2, Y, L	
q_3				q_3, Y, R	$q_{\text{accept}}, ⊔, \mathrm{R}$

- Una línea por casilla llena: `estado lee -> destino escribe movimiento`. El blanco es `_`.
- 11 transiciones de 20. Una casilla vacía: la entrada ya no puede estar en el lenguaje.
- El archivo real empieza con un comentario.

SECCIÓN 03

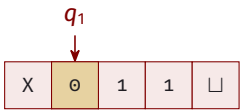
Configuraciones

03

Configuraciones

Configuración *(configuration)*: $u\ q\ v$

El estado es q, la cinta tiene uv seguido de blancos, y el cabezal está sobre el **primer símbolo de** v.



se escribe $X\ q_1\ 011$

- **Inicial** con la entrada w: $q_0\, w$. Con $w = \varepsilon$: $q_0\, \sqcup$.
- **De aceptación**: el estado es q_{accept}. **De rechazo**: es q_{reject}. Las dos son **finales**.
- $u\ q\ v$ y $u\ q\ v \sqcup$ son la misma: v se escribe hasta el último símbolo no blanco.
- Error común: escribir el estado *después* del símbolo que se lee. Va justo antes.

Una configuración produce la siguiente

Para $a, b, c \in \Gamma$ y $u, v \in \Gamma^*$:

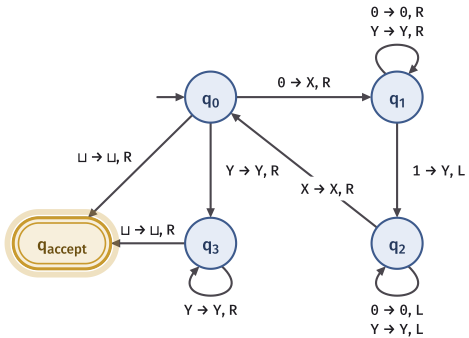
$u\ q\ bv$	produce	$uc\ r\ v$	si $\delta(q, b) = (r, c, \mathrm{R})$
$ua\ q\ bv$	produce	$u\ r\ acv$	si $\delta(q, b) = (r, c, \mathrm{L})$
$q\ bv$	produce	$r\ cv$	si $\delta(q, b) = (r, c, \mathrm{L})$: el borde izquierdo

- A la derecha: el símbolo escrito queda a la izquierda del estado.
- A la izquierda: el estado retrocede un símbolo. En la primera celda, no se mueve.

La ejecución de M con w

$C_0, C_1, C_2, \ldots$ con $C_0 = q_0\, w$ y cada C_i produce C_{i+1}. Es única. Termina en una configuración final, o no termina.

La máquina en marcha con 0011



LA CINTA



CONFIGURACIONES como en `run_tm`

0	q_0 0011
1	X q_1 011
2	X0 q_1 11
3	X q_2 0Y1
4	q_2 X0Y1
5	X q_0 0Y1
6	XX q_1 Y1
7	XXY q_1 1
8	XX q_2 YY
9	X q_2 XYY
10	XX q_0 YY
11	XXY q_3 Y
12	XXYY q_3 ⊔
13	XXYY⊔ q_{accept} ⊔

Paso 13: $\delta(q_3, \sqcup) = (q_{accept}, \sqcup, R)$

Lee el primer blanco: todo quedó tachado. Acepta, en 13 pasos.

La ejecución con 0011

terminal

```
$ build/run_tm machines/zeros_ones.tm 0011
w = 0011
    0  q0 0011
    1  X q1 011
    2  X0 q1 11
    3  X q2 0Y1
    4  q2 X0Y1
    5  X q0 0Y1
    6  XX q1 Y1
    7  XXY q1 1
    8  XX q2 YY
    9  X q2 XYY
   10  XX q0 YY
   11  XXY q3 Y
   12  XXYY q3 _
   13  XXYY_ q_accept _
  acepta en 13 pasos
```

- 0 a 1: en q_0 lee 0. La tabla dice (q_1, X, R).
- 2 a 3: en q_1 lee 1. La tabla dice (q_2, Y, L): el estado aparece un símbolo más a la izquierda.
- 3 a 5: vuelve hasta la X y da un paso a la derecha. **Terminó la primera vuelta.**
- 5 a 10: la segunda vuelta.
- 10: q_0 ya no encuentra un 0 sino una Y. Pasa a q_3, que recorre las Y.
- 12: lee el primer blanco y acepta.

Los programas escriben _ por ␣, con espacios alrededor del estado.

En C++: una configuración y un paso

tm.h

```cpp
struct Config {
    std::string state;
    std::string cells;   // las celdas ya visitadas
    std::size_t head = 0;
};
inline void apply(Config& c, const Action& action) {
    if (c.head >= c.cells.size()) {
        c.cells.resize(c.head + 1, BLANK);
    }
    c.cells[c.head] = action.write;
    c.state = action.state;
    if (action.move == 'R') {
        c.head++;
    } else if (c.head > 0) {
        c.head--;  // en la primera celda, L no mueve
    }
}
```

- El `struct Tm` tiene los siete componentes, con los nombres del archivo.
- La cinta infinita es un `std::string` finito: lo que no está, es blanco.
- Crece solo cuando el cabezal llega a una celda nueva.
- El borde izquierdo es el último `if`.

SECCIÓN 04

Aceptar, rechazar o no detenerse

04

Tres maneras de terminar

Aceptar, rechazar, reconocer

M **acepta** w si su ejecución llega a una configuración de aceptación; la **rechaza** si llega a una de rechazo.

$L(M) = \{ w \in \Sigma^* : M \text{ acepta } w \}$, y M **reconoce** $L(M)$.

Con una entrada, la máquina:

1. llega a q_{accept} y **acepta**;
2. llega a q_{reject} y **rechaza**;
3. **no se detiene nunca**: la ejecución es infinita.

Un DFA lee un símbolo por paso y la entrada se acaba. Aquí nada obliga a la máquina a avanzar hacia un final.

Una máquina que no se detiene

has_one_loops.tm

```
states: q0 q_accept q_reject
alphabet: 0 1
tape: 0 1 _
start: q0
accept: q_accept
reject: q_reject
delta:
q0 0 -> q0 0 R
q0 1 -> q_accept 1 R
q0 _ -> q0 _ R
```

Busca un 1 hacia la derecha, y no deja de buscar cuando la entrada se acaba.

terminal

```
$ build/run_tm --limit 1000000 --show 6 \
      machines/has_one_loops.tm 00 001
w = 00
    0  q0 00
    1  0 q0 0
    2  00 q0 _
    3  00_ q0 _
    4  00__ q0 _
    5  00___ q0 _
   ...
  no se detuvo en 1000000 pasos
w = 001
    0  q0 001
    1  0 q0 01
    2  00 q0 1
    3  001 q_accept _
  acepta en 3 pasos
```

“No se detuvo en N pasos” no es “no se detiene”

- Un simulador da a lo sumo N pasos. Un millón de pasos es un hecho sobre un millón de pasos.
- Que no se detiene **se demuestra**: con $w = 0^n$ y $t \geq n$, después de t pasos está en $0^n \sqcup^{t-n} q_0 \sqcup$. Por inducción sobre t.
- Ahí **ninguna configuración se repite**. “No detenerse” no es “dar vueltas”.
- A la derecha sí se repite: cada paso es un L en la primera celda.
- El `TLE` del juez es esta respuesta: no sabe si hay un ciclo infinito o si el programa era lento.

terminal

```
$ build/run_tm --limit 8 \
      machines/left_end.tm 01
w = 01
    0  q0 01
    1  0 q0 1
    2  01 q0 _
    3  0 q1 1
    4  q1 01
    5  q1 01
    6  q1 01
    7  q1 01
    8  q1 01
  no se detuvo en 8 pasos
```

SECCIÓN 05

Tres máquinas, paso a paso

05

$\{0^n1^n\}$: lo que conserva cada vuelta

Vuelta: de q_0 leyendo un 0 hasta la siguiente vez que entra a q_0.

Afirmación: después de i vueltas completas, está en q_0, en la celda i, y

1. las celdas $0, \ldots, i-1$ tienen X, y en w tenían 0;
2. exactamente i de las demás celdas tienen Y, y en w tenían 1;
3. todas las otras están como en w.

- Por inducción sobre i: cada vuelta tacha un 0 y un 1 de w, y vuelve a la X que acaba de escribir.
- **Acepta desde** q_3: las celdas $i, \ldots, n-1$ son todas Y. Por (2) son i: $n = 2i$ y $w = 0^i1^i$.
- **Acepta desde** q_0: lee un blanco en la celda i; todo es X y no hay sitio para las Y: $w = \varepsilon$.
- **Si** $w = 0^n1^n$: después de i vueltas está en $X^i\ q_0\ 0^{n-i}\ Y^i\ 1^{n-i}$, y al final acepta.

$\{0^n1^n\}$: se detiene siempre, y cuánto tarda

Con una entrada de longitud n:

- Cada vuelta tacha dos celdas: a lo sumo $n/2$ vueltas.
- Una vuelta da a lo sumo $2n - 1$ pasos.
- El tramo final solo va a la derecha: $\leq n + 2$.

$$\tfrac{n}{2}(2n - 1) + n + 2 \;\leq\; n^2 + n + 2 \text{ pasos}$$

Decide: acepta solo cadenas del lenguaje, las acepta todas y se detiene siempre. Son tres cosas que demostrar.

terminal

```
$ build/run_tm --quiet \
      machines/zeros_ones.tm \
      01 0011 000111 00001111
w = 01
  acepta en 5 pasos
w = 0011
  acepta en 13 pasos
w = 000111
  acepta en 25 pasos
w = 00001111
  acepta en 41 pasos
```

Con 0^n1^n: n vueltas de $2n + 1$ pasos, más $n + 1$: $2n^2 + 2n + 1$.

$\{w\#w\}$: llevar un símbolo por viaje

- Comparar el primero de la izquierda con el primero de la derecha, después el segundo…
- No puede recordar posiciones: **tacha** lo que ya comparó.
- No puede recordar una cadena: lleva **un símbolo**, guardado en el estado.

q_0	tacha el primero sin tachar y recuerda cuál era
q_1, q_2	lleva un 0 / un 1 hasta el separador
q_3, q_4	salta las X y compara; si coincide, tacha
q_5	vuelve sobre las X hasta el separador
q_6	sigue hasta la última X de la izquierda
q_7	la izquierda se acabó: a la derecha solo deben quedar X

El separador en los archivos

En la teoría es $\#$. En los archivos, `#` empieza un comentario: ahí el separador es `=`.

$\{w\#w\}$: la tabla

δ	0	1	=	X	$\sqcup$
q_0	q_1, X, R	q_2, X, R	$q_7, =, \mathrm{R}$		
q_1	$q_1, 0, \mathrm{R}$	$q_1, 1, \mathrm{R}$	$q_3, =, \mathrm{R}$		
q_2	$q_2, 0, \mathrm{R}$	$q_2, 1, \mathrm{R}$	$q_4, =, \mathrm{R}$		
q_3	q_5, X, L			q_3, X, R	
q_4		q_5, X, L		q_4, X, R	
q_5			$q_6, =, \mathrm{L}$	q_5, X, L	
q_6	$q_6, 0, \mathrm{L}$	$q_6, 1, \mathrm{L}$		q_0, X, R	
q_7				q_7, X, R	$q_{\mathrm{accept}}, \sqcup, \mathrm{R}$

- 20 transiciones escritas de 40. Las casillas vacías hacen buena parte del trabajo.
- q_3 con 1: la comparación que falla. q_0 con $\sqcup$: la cadena vacía no tiene separador.
- Con $w\#w$ y $|w| = n$: $2n^2 + 4n + 2$ pasos. Con cualquier entrada de longitud n: a lo sumo $n^2 + n + 2$.

$\{w\#w\}$: la ejecución con `01=01`

```
terminal
$ build/run_tm machines/w_sep_w.tm 01=01
w = 01=01
    0  q0 01=01
    1  X q1 1=01
    2  X1 q1 =01
    3  X1= q3 01
    4  X1 q5 =X1
    5  X q6 1=X1
    6  q6 X1=X1
    7  X q0 1=X1
    8  XX q2 =X1
    9  XX= q4 X1
```

```
sigue
   10  XX=X q4 1
   11  XX= q5 XX
   12  XX q5 =XX
   13  X q6 X=XX
   14  XX q0 =XX
   15  XX= q7 XX
   16  XX=X q7 X
   17  XX=XX q7 _
   18  XX=XX_ q_accept _
acepta en 18 pasos
```

- 0 a 7: lleva el 0 en q_1 y lo compara en q_3.
- 7 a 14: lleva el 1 en q_2 y q_4.
- 9: q_4 salta la X de la vuelta anterior.
- 14: la izquierda se acabó. q_7 revisa la derecha.

$\{a^n b^n c^n\}$: el lenguaje que no era libre de contexto

- En cada vuelta: tachar una a, una b y una c.
- Tachar **no destruye**: las marcas de las b siguen ahí cuando llega el turno de las c.
- El **orden** lo vigilan las casillas vacías.
- Las **cantidades**, el final: q_4 y q_5 no admiten nada sin tachar.

δ	a	b	c	X	Y	Z	$\sqcup$
q_0	q_1, X, R				q_4, Y, R		q_{accept}
q_1	q_1, a, R	q_2, Y, R			q_1, Y, R		
q_2		q_2, b, R	q_3, Z, L			q_2, Z, R	
q_3	q_3, a, L	q_3, b, L		q_0, X, R	q_3, Y, L	q_3, Z, L	
q_4					q_4, Y, R	q_5, Z, R	
q_5						q_5, Z, R	q_{accept}

18 transiciones de 42. Con $a^n b^n c^n$: $4n^2 + 3n + 1$ pasos.

Marcar y volver: la misma técnica de $\{0^n 1^n\}$. Al autómata de pila le faltaba volver sobre lo escrito sin borrarlo.

$\{a^n b^n c^n\}$: la ejecución con aabbcc

terminal

```
$ build/run_tm machines/anbncn.tm aabbcc
w = aabbcc
    0  q0 aabbcc
    1  X q1 abbcc
    2  Xa q1 bbcc
    3  XaY q2 bcc
    4  XaYb q2 cc
    5  XaY q3 bZc
    6  Xa q3 YbZc
    7  X q3 aYbZc
    8  q3 XaYbZc
    9  X q0 aYbZc
   10  XX q1 YbZc
   11  XXY q1 bZc
```

sigue

```
   12  XXYY q2 Zc
   13  XXYYZ q2 c
   14  XXYY q3 ZZ
   15  XXY q3 YZZ
   16  XX q3 YYZZ
   17  X q3 XYYZZ
   18  XX q0 YYZZ
   19  XXY q4 YZZ
   20  XXYY q4 ZZ
   21  XXYYZ q5 Z
   22  XXYYZZ q5 _
   23  XXYYZZ_ q_accept _
acepta en 23 pasos
```

- 0 a 9: primera vuelta.
- 9 a 18: segunda.
- 18 a 23: la comprobación final.

Las pruebas comparan cada máquina con una función de C++ en todas las cadenas cortas.

SECCIÓN 06

Reconocer y decidir

06

Reconocer y decidir

Decididor *(decider)*, reconocible, decidible

Un **decididor** es una máquina que se detiene con *toda* entrada. Un decididor M **decide** $L(M)$.

L es **reconocible** si alguna máquina lo reconoce, y **decidible** si alguna máquina lo decide.

	con $w \in L$	con $w \notin L$
M **reconoce** L	acepta	rechaza o no se detiene
M **decide** L	acepta	rechaza

- Todo lenguaje decidible es reconocible. La otra dirección es la semana 9.
- Con una máquina que solo reconoce, quien lleva una hora esperando no sabe si la respuesta es no.

El mismo lenguaje, dos máquinas

$A = \{ w : w \text{ tiene algún } 1 \}$

- `has_one_loops.tm` lo reconoce y **no** lo decide: con 00 no se detiene.
- `has_one.tm`: en el primer blanco, rechaza. **Lo decide.**
- A es decidible: basta que *exista* un decididor.

terminal

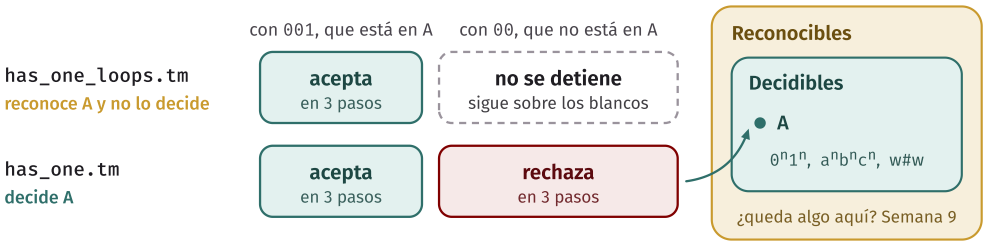
```
$ build/run_tm --quiet \
      machines/has_one.tm 000 0010
w = 000
  rechaza en 4 pasos
w = 0010
  acepta en 3 pasos
```

Error común

Decir que una máquina "decide" cuando hay entradas con las que no se detiene. *Decidir* habla de **todas** las entradas. Y que *una* máquina no decida L no hace a L indecidible.

Reconocer y decidir, con las dos máquinas

A = { w : w tiene algún 1 }, y dos máquinas con L(M) = A



Reconocer A es aceptar exactamente las cadenas de A. Decidirlo es, además, detenerse con toda entrada.

A es decidible porque existe un decididor, aunque otra máquina que lo reconoce no se detenga con 00.

Dos consecuencias de las definiciones

Todo lenguaje regular es decidible

Dado un DFA B, la máquina tiene sus estados y solo va a la derecha: $\delta(q, a) = (\delta_B(q, a), a, \mathrm{R})$. En el primer blanco acepta si $q \in F$, y si no, rechaza. Se detiene en $n + 1$ pasos.

Los decidibles son cerrados bajo complemento

Si M decide L, intercambiar q_{accept} y q_{reject} da una máquina que decide el complemento.

- Con una máquina que solo reconoce, no sirve: las entradas con las que no se detiene siguen sin respuesta. Como el complemento de un DFA sin estado trampa.
- Semana 9: los reconocibles **no** son cerrados bajo complemento.

Tres niveles de descripción, para la misma máquina

De implementación: qué hace el cabezal, en prosa.

nivel de implementación

```
M = "Con la entrada w:
  1. Si el cabezal lee el separador, ir a 5.
     Si lee un blanco, rechazar. Si no,
     recordar el símbolo y tacharlo.
  2. Avanzar hasta el separador. Si antes
     aparece un blanco, rechazar.
  3. Avanzar sobre lo tachado hasta el primer
     símbolo sin tachar. Si es el recordado,
     tacharlo. Si no, rechazar.
  4. Retroceder hasta el último tachado de la
     izquierda, avanzar una celda, ir a 1.
  5. Avanzar sobre lo tachado. Si después
     viene un blanco, aceptar. Si no, rechazar."
```

Formal: la tabla de δ, o el archivo. 20 transiciones.

Alto: un algoritmo en pasos, sin cinta ni cabezal.

nivel alto

```
M = "Con la entrada w:
  1. Comprobar que w tiene exactamente
     un separador. Si no, rechazar.
  2. Comparar la parte anterior al
     separador con la posterior.
     Si son iguales, aceptar.
     Si no, rechazar."
```

Cuándo se usa cada nivel

Nivel	Qué se escribe	Para qué
Formal	La tabla o el archivo	Ejecutarla, programarla
De implementación	El cabezal y las marcas	Ver que cabe en el modelo; contar pasos
Alto	Un algoritmo en pasos	Todo lo demás, desde la semana 9

- En el nivel alto, cada paso tiene que ser claramente realizable con una cantidad finita de trabajo.
- Si se afirma que la máquina **decide**, tiene que verse que termina.
- ¿Con qué derecho se salta del primero al tercero? Es la sesión B.

Como en cualquier software

Código de máquina, código fuente, pseudocódigo. Se confía en el nivel alto porque hay un compilador que lo baja.

SECCIÓN 07

El modelo es robusto

07

El modelo es robusto

¿Por qué una sola cinta? ¿Por qué el cabezal no puede quedarse quieto?
Robusto: las variantes razonables del modelo reconocen exactamente los mismos lenguajes.

> **Cómo se demuestra: simular**
>
> Dada una máquina cualquiera de un modelo, se construye una del otro que reconoce el mismo lenguaje. Una dirección suele ser trivial.

Calentamiento: permitir S, quedarse quieto.

- $\delta(q, a) = (r, b, \mathrm{S})$ se cambia por dos pasos: escribir b e ir a la derecha, a un estado nuevo r'; y desde r', volver a la izquierda y pasar a r.
- Después de un R el cabezal no está en la primera celda: el L lo devuelve.
- El mismo lenguaje, con el doble de pasos como mucho.

SECCIÓN 08

Máquinas de varias cintas

08

Máquinas de varias cintas

Máquina de k cintas *(multitape Turing machine)*

k cintas, cada una con su cabezal. La entrada va en la primera.
$\delta : Q' \times \Gamma^k \to Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k$

$\{0^n 1^n\}$ con dos cintas, sin ir y venir:

Estado	Lee	Escribe	Mueve	Pasa a	
q_0	$\sqcup, \sqcup$	$\sqcup, \sqcup$	S, S	q_{accept}	la entrada vacía
q_0	0, $\sqcup$	0, \$	S, R	q_1	marca el inicio de la cinta 2
q_1	0, $\sqcup$	0, 0	R, R	q_1	copia un cero
q_1	1, $\sqcup$	1, $\sqcup$	S, L	q_2	la cinta 2 vuelve al último cero
q_2	1, 0	1, 0	R, L	q_2	un uno por cada cero
q_2	$\sqcup$, \$	$\sqcup$, \$	S, S	q_{accept}	se acabaron a la vez

Con $0^n 1^n$: $2n + 3$ pasos. Con una cinta eran $2n^2 + 2n + 1$.

Dos cintas, ejecutadas

terminal

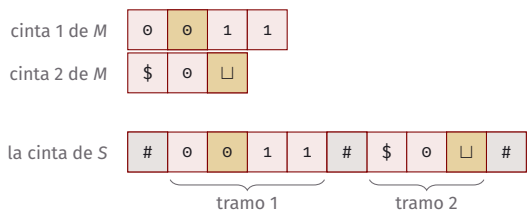
```
$ build/multitape_demo run 0011
w = 0011
  0  q0   cinta 1: [0]011   cinta 2: [_]
  1  q1   cinta 1: [0]011   cinta 2: $[_]
  2  q1   cinta 1: 0[0]11   cinta 2: $0[_]
  3  q1   cinta 1: 00[1]1   cinta 2: $00[_]
  4  q2   cinta 1: 00[1]1   cinta 2: $0[0]
  5  q2   cinta 1: 001[1]   cinta 2: $[0]0
  6  q2   cinta 1: 0011[_]   cinta 2: [$]00
  7  q_accept   cinta 1: 0011[_]   cinta 2: [$]00
  acepta en 7 pasos
```

- Entre corchetes, el símbolo bajo cada cabezal. La máquina está en `multitape.h`.
- Más cómoda y más rápida. ¿Más **poderosa**?

Una cinta simula a k

Teorema

Para toda máquina de varias cintas hay una de una cinta que reconoce el mismo lenguaje.



- S guarda las k cintas una tras otra, separadas por #. El símbolo bajo cada cabezal lleva una **marca**.

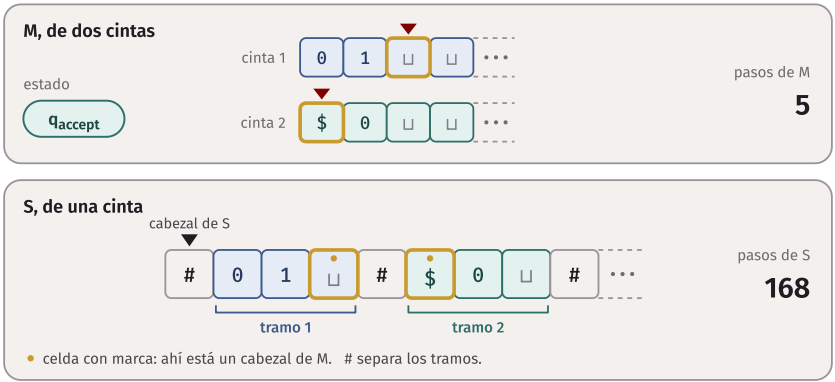
Cómo trabaja S

1. **Preparar la cinta:** correr w una celda y dejar $\#\,\dot{w}_1 w_2 \cdots w_n\,\#\,\dot{\sqcup}\,\#\cdots\#$, con k tramos.
2. **Para simular un paso de M:**
   - 2.1 Recorrer la cinta anotando el símbolo marcado de cada tramo. Volver.
   - 2.2 Con el estado de M y los k símbolos, buscar en δ qué hace M.
   - 2.3 Recorrerla otra vez: en cada tramo, escribir el símbolo nuevo y mover la marca una celda.
     - Si la marca saldría por la izquierda del tramo: se queda. *Es el borde izquierdo de M.*
     - Si caería sobre el separador de la derecha: poner un blanco con marca y correr todo lo que sigue una celda.
   - 2.4 Volver. Si M llegó a un estado final, hacer lo mismo. Si no, repetir.

Por qué cabe en una máquina

Fuera de la cinta, S recuerda el estado de M, k símbolos, la acción y cuántos separadores pasó: finitas combinaciones. Cada una es un estado de S.

Una cinta simula dos, paso a paso



Paso 5 de M: lee ␣ y $: los unos y los ceros se acabaron a la vez, y M acepta.

S recorre su cinta, encuentra en δ que M pasa a q_{accept}, y acepta. Los 5 pasos de M le costaron 168.

La simulación, ejecutada

terminal

```
$ build/multitape_demo sim 01
w = 01
pasos de M   estado de M   cinta de S              pasos de S
         0   q0            #[0]1#[_]#                      12
         1   q1            #[0]1#$[_]#                     36
         2   q1            #0[1]#$0[_]#                    64
         3   q2            #0[1]#$[0]_#                    94
         4   q2            #01[_]#[$]0_#                  136
         5   q_accept      #01[_]#[$]0_#                  168
  acepta: 5 pasos de M, 168 pasos de S (12 para preparar la cinta)
```

- **Invariante**: después de simular t pasos, la cinta de S representa la configuración de M después de t pasos. Entonces $L(S) = L(M)$.
- 3 a 4: crece el primer tramo, y se corre todo el segundo. Las pruebas revisan el invariante en cada paso.

Cuánto cuesta: la cuenta

M de k cintas se detiene en t pasos con una entrada de longitud n. ℓ: celdas en uso en la cinta de S.

- Al empezar, $\ell \leq n + 2k + 1$.
- Cada paso simulado agrega a lo sumo k celdas: $\ell \leq n + 2k + 1 + kt$.
- t pasos, cada uno de costo $\leq (2k + 6)\ell$.
- El cuadrado: simular *un* paso obliga a recorrer *toda* la cinta, que mide tanto como pasos lleva M.

Simular un paso	**a lo sumo**
ir al final y volver	2ℓ
avanzar actualizando	ℓ
mover marcas a la izquierda	$2k$
correr la cinta, si un tramo crece	$2k\ell$
volver	ℓ
total	$(2k + 6)\ell$

Teorema

S se detiene en a lo sumo $2(n + 2k + 1) + t\,(2k + 6)(n + 2k + 1 + kt)$ pasos. Si $t \geq n$: $O(t^2)$. Vuelve en la semana 12.

Cuánto cuesta: medido

terminal

```
$ build/multitape_demo table
Entrada 0^n 1^n, de longitud 2n. Pasos hasta aceptar:

   n   una cinta (zeros_ones.tm)   dos cintas   dos cintas simuladas en una
   5                          61           13                           992
  10                         221           23                          3012
  20                         841           43                         10352
  40                        3281           83                         38232
  80                       12961          163                        146792
 160                       51521          323                        575112
 320                      205441          643                       2276552
```

- Dos cintas: $2n + 3$. Simulada: al duplicar n se multiplica por casi 4. **Cuadrática.**
- La máquina pensada para una cinta también lo es, con una constante unas once veces menor.

SECCIÓN 09

Máquinas no deterministas

09

Máquinas no deterministas

No determinista *(nondeterministic)*

$\delta : Q' \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})$

$\delta(q, a)$ es un *conjunto* de acciones posibles.

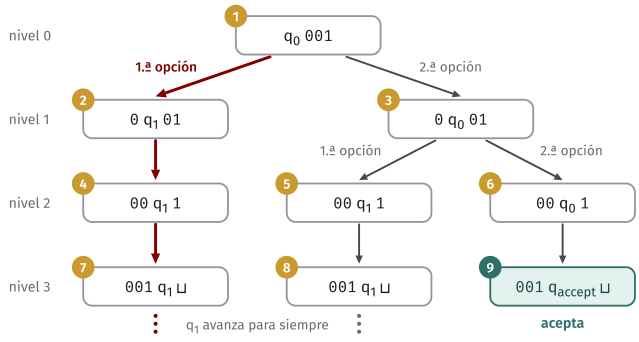
- Las ejecuciones con w forman un **árbol**: la raíz es $q_0\, w$, y los hijos de una configuración son las que produce.
- N **acepta** w si alguna rama llega a q_{accept}. No importa cuántas rechacen ni cuántas no se detengan.
- NFA: no ganaba poder. PDA: sí. ¿Y aquí?

nondet_branch.tm

```
states: q0 q1 q_accept q_reject
alphabet: 0 1
tape: 0 1 _
start: q0
accept: q_accept
reject: q_reject
delta:
q0 0 -> q1 0 R
q0 0 -> q0 0 R
q0 1 -> q_accept 1 R
q1 0 -> q1 0 R
q1 1 -> q1 1 R
q1 _ -> q1 _ R
```

Dos líneas para $(q_0, 0)$. q_1 avanza para siempre.

El árbol de ejecuciones con 001



En profundidad: sigue siempre la primera opción, entra en la rama de q_1 y no vuelve nunca.

n Por niveles: el orden en que D revisa las configuraciones. La novena es de aceptación.

La configuración 00 q_1 1 aparece dos veces: se llega a ella por dos ramas, y el árbol las cuenta por separado.

El mismo árbol, nivel por nivel

terminal

```
$ build/ntm_demo machines/nondet_branch.tm 001
w = 001
nivel 0:   q0 001
nivel 1:   0 q1 01   0 q0 01
nivel 2:   00 q1 1   00 q1 1   00 q0 1
nivel 3:   001 q1 _   001 q1 _   001 q_accept _
por niveles: acepta en el nivel 3, después de revisar 9 configuraciones
siempre la primera opción: no se detuvo en 1000000 pasos
```

- En el nivel *d* están las configuraciones a las que se llega con *d* pasos.
- Una rama acepta: la máquina acepta 001, aunque las otras dos sigan sin fin.
- La misma configuración puede aparecer en dos ramas: el árbol las cuenta por separado.

En profundidad no sirve. Por niveles, sí

Teorema

Para toda máquina no determinista hay una determinista que reconoce el mismo lenguaje.

En profundidad: el intento que falla

Seguir una rama hasta el final y después volver. Pero una rama puede **no tener final**: *D* entra y no vuelve.

Por niveles *(breadth-first)*

Primero todas las configuraciones a un paso de la raíz, después las que están a dos… Una rama infinita ya no estorba.

Cambiar el orden de las opciones no lo arregla: para cualquier orden fijo hay una máquina cuya rama infinita está primero.

La máquina D, y por qué funciona

nivel alto

```
D = "Con la entrada w:
  1. Poner en la cola la configuración
     inicial de N con w.
  2. Mientras la cola no esté vacía:
     a. Sacar la configuración C del frente.
     b. Si C es de aceptación, aceptar.
     c. Por cada acción posible en C, agregar
        al final la configuración que resulta.
  3. Rechazar."
```

La cola es una cinta: configuraciones $u\,q\,v$ separadas por #. Varias cintas ya se reducen a una.

- **Si D acepta, N acepta:** todo lo que entra a la cola está en el árbol de N.
- **Si N acepta, D acepta:** cada configuración tiene a lo sumo b hijos, el nivel d tiene a lo sumo b^d, y la cola respeta los niveles. Antes de la aceptación del nivel d hay a lo sumo

$$1 + b + \cdots + b^d$$

configuraciones: una cantidad **finita**.

- Si todas las ramas se detienen, el árbol es finito, la cola se vacía y D rechaza: D es un decididor.

El costo del no determinismo

- Si la rama que acepta tiene t pasos, D puede procesar del orden de b^t configuraciones: tiempo exponencial. No se conoce ninguna simulación determinista que lo evite.

Variante	¿Agrega poder?	Costo de simularla
Quedarse quieto	No	el doble de pasos
Varias cintas	No	$O(t^2)$
No determinismo	No	exponencial, hasta donde se sabe

Error común

"La máquina no determinista prueba todas las ramas." No recorre nada: acepta si *existe* una rama que acepta. Quien recorre el árbol es la determinista, y paga por cada configuración.

Semana 12: la diferencia entre el cuadrado y la exponencial es la pregunta de si P es igual a NP.

SECCIÓN 10

Enumeradores

10



Enumeradores

Enumerador *(enumerator)*

Una máquina de Turing con una **impresora**. Empieza con la cinta en blanco y va imprimiendo cadenas.
El lenguaje que **enumera**: todas las cadenas que imprime en algún momento.

Sin orden fijo, puede repetir, y puede no detenerse nunca.

Teorema

Un lenguaje es reconocible si y solo si algún enumerador lo enumera.

- **De enumerador a máquina:** con la entrada w, ejecutar E y comparar con w cada cadena que imprime. Si alguna es igual, aceptar.
- De ahí el otro nombre de los reconocibles: **recursivamente enumerables**.

De una máquina que reconoce a un enumerador

El intento que falla: ejecutar M con s_1, después con s_2… Si M no se detiene con una cadena, nunca llega a las siguientes.

nivel alto

```
E = "Sin mirar la entrada:
  1. Para i = 1, 2, 3,..:
  2.    Ejecutar M durante i pasos con cada
        una de las cadenas s1, s2,.., si.
  3.    Imprimir las que M haya aceptado."
```

- Cada ronda termina: i ejecuciones de i pasos.
- Si M acepta s_j en t pasos, sale en la ronda $\max(j, t)$.

terminal

```
$ build/enumerate machines/has_one_loops.tm 8
ronda 1:
ronda 2:
ronda 3: 1
ronda 4: 1
ronda 5: 1 01
ronda 6: 1 01 10
ronda 7: 1 01 10 11
ronda 8: 1 01 10 11
```

Las primeras líneas de la salida. Con ε, 0 y 00 la máquina no se detiene, y el enumerador no se atasca.

SECCIÓN 11

La tesis de Church-Turing

11



Otros modelos, el mismo poder

Propuestos por separado, en los mismos años, para la misma pregunta: ¿qué es "calculable"?

- **Cálculo** λ (Church): solo funciones que se aplican a funciones.
- **Funciones recursivas** (Gödel, Herbrand, Kleene): composición, recursión y búsqueda.
- **Máquinas de Turing** (1936): una persona que calcula con lápiz y papel.
- **Máquina RAM**: registros y memoria con direcciones. El modelo de Algoritmos.
- **Cualquier lenguaje de programación**, con memoria sin límite.

Cada equivalencia es un teorema

`tm.h` es un programa en C++ que ejecuta cualquier máquina de Turing. Al revés: C++ se compila a instrucciones sobre registros y memoria, y una máquina de varias cintas guarda la memoria en una cinta. Más lento. No menos.

La velocidad no es poder de cómputo

Error común

Creer que una computadora real calcula más que una máquina de Turing porque es más rápida.

- La pregunta de esta parte es qué problemas **tienen solución**.
- Una máquina más rápida resuelve los mismos problemas en menos tiempo. Ni uno más.
- Si se compara con cuidado, la que pierde es la computadora real: su memoria es **finita**. En rigor es un autómata finito con una cantidad enorme de estados.
- La máquina de Turing es el modelo de una computadora a la que nunca se le acaba la memoria.

La tesis de Church-Turing

Tesis de Church-Turing *(Church-Turing thesis)*

Lo que se puede calcular con un procedimiento mecánico es exactamente lo que se puede calcular con una máquina de Turing.

- **Procedimiento mecánico**: una receta finita y precisa, que se sigue con lápiz y papel, sin ingenio.
- La tesis afirma que ningún procedimiento mecánico queda fuera.

No es un teorema

Relaciona un objeto matemático con una noción *sin* definición matemática. Es una afirmación **sobre el mundo**: se sostiene con evidencia, y se podría refutar.

La evidencia

1. **Todos los modelos coinciden.** Ideas muy distintas trazaron la misma frontera.
2. **El modelo es robusto.** Más cintas, no determinismo, otro movimiento: nada cambia. Lo de hoy.
3. **El análisis de Turing.** Quien calcula a mano distingue finitos símbolos, tiene finitos "estados mentales" y mira una parte acotada del papel. Eso es una máquina de Turing.
4. **No hay contraejemplo.** En casi noventa años, todo algoritmo inventado se pudo programar. Una computadora cuántica calcula las mismas funciones; lo que podría cambiar es cuánto tarda.

Error común: "por el *teorema* de Church-Turing…"

Que varias cintas equivalen a una es un teorema. Que *todo* algoritmo es una máquina de Turing es la tesis.

Qué permite la tesis

Una definición

Algoritmo = máquina de Turing que se detiene con toda entrada.

- Para mostrar que un problema *tiene* algoritmo, basta escribirlo.
- Para demostrar que *no tiene*, hay que razonar sobre todos los algoritmos a la vez.

Un permiso

Describir las máquinas en el nivel alto, con cualquier comodidad.

- Cada vez, se afirma que existe una tabla que lo hace.
- Desde la semana 9 casi no escribiremos tablas.

Hilbert, 1900: pidió "un proceso" que decidiera si un polinomio con coeficientes enteros tiene una raíz entera. En 1970 Matiyasevich demostró que no existe. En 1900 esa demostración no se podía ni empezar.

El nivel alto, y otra vez reconocer y decidir

$D_1 = \{ \langle p \rangle : p \text{ es un polinomio en } x \text{ con coeficientes enteros y alguna raíz entera} \}$

reconoce

```
M1 = "Con la entrada ⟨p⟩:
  1. Evaluar p en x = 0, 1, −1, 2, −2,...
  2. Si alguna vez da 0, aceptar."
```

decide

```
M2 = "Con la entrada ⟨p⟩:
  1. Si a0 = 0, aceptar: x = 0 es raíz.
  2. Evaluar p en cada entero x con
     1 ≤ |x| ≤ |a0|.
  3. Si alguno da 0, aceptar.
     Si no, rechazar."
```

- Si no hay raíz entera, no se detiene.
- Si $r \neq 0$ es raíz de $a_d x^d + \cdots + a_1 x + a_0$, entonces $a_0 = -r(a_d r^{d-1} + \cdots + a_1)$: r divide a a_0.
- Para decidir hace falta saber **cuándo dejar de buscar**. Aquí, la cota $|r| \leq |a_0|$.
- Con varias variables el lenguaje sigue siendo reconocible, y no es decidible.

Codificar objetos como cadenas

$\langle O \rangle$: codificación *(encoding)*

Una cadena que representa al objeto O. Varios objetos: $\langle O_1, \ldots, O_k \rangle$.

- Un grafo, como lista de nodos y de aristas:
  `(1,2,3,4)((1,2),(2,3),(3,1),(1,4))`
- La máquina comprueba primero que la cadena codifica lo esperado. Si no, rechaza.
- Para la computabilidad, los detalles no importan. Para el costo, sí: semana 12.

nivel alto

```
M = "Con la entrada ⟨G⟩, un grafo no
     dirigido con al menos un nodo:
  1. Marcar el primer nodo de G.
  2. Repetir hasta que no se marque
     ningún nodo nuevo:
  3.   Por cada nodo, marcarlo si tiene
       una arista a un nodo marcado.
  4. Si todos quedaron marcados,
     aceptar. Si no, rechazar."
```

Decide si G es conexo: a lo sumo tantas repeticiones como nodos, más una.

Una máquina también es una cadena

- Una máquina de Turing es una séptupla de conjuntos finitos y una tabla finita: se puede escribir completa.
- El archivo `machines/zeros_ones.tm`, de 327 caracteres, es un $\langle M \rangle$.
- Entonces una máquina puede recibir como entrada **otra máquina**: $\langle M, w \rangle$.
- Es el laboratorio de hoy: su programa hace con $\langle M, w \rangle$ lo que M haría con w.

En la práctica

Un compilador, un intérprete, el juez del curso: programas que reciben programas. Que los programas sean datos hizo posible la computadora de propósito general.

Semana 9: ¿qué se puede averiguar sobre una máquina leyendo su descripción?

Laboratorio: un simulador de máquinas de Turing

1. Leer una máquina de una cinta, un límite N y una lista de cadenas.
2. Responder `acepta`, `rechaza` o `no se detuvo en N pasos`.
3. Guardar la cinta de modo que crezca a la derecha sin límite, con pasos de tiempo constante.
4. Con `–trace`, imprimir las configuraciones $u\,q\,v$. El juez no lo usa.
5. Escribir la máquina de $\{0^n1^n\}$ y comprobarla con su simulador.

Dónde: `code/week-07/lab/tm_sim`.
Bordes: la cadena vacía, un `L` en la primera celda, una transición que falta, una máquina que no se detiene, y una que se detiene justo en el paso N.

`casos/01.in`, el final

```
q3 Y -> q3 Y R
q3 _ -> q_accept _ R
limit: 40
strings: 4
0011

010
00001111
```

Salida: `acepta`, `acepta`, `rechaza`, `no se detuvo en 40 pasos`.

Para la próxima clase

- Leer los apuntes de la semana 7 y terminar el laboratorio.
- Preguntas de repaso de la guía: las configuraciones de la máquina de $\{0^n 1^n\}$ con 0011; una descripción de implementación para $\{a^i b^j c^k : i \times j = k\}$; por qué el recorrido va por niveles; qué evidencia sostiene la tesis.
- Lo que sigue en computabilidad, en la semana 9: lo decidible y lo indecidible. Máquinas que reciben máquinas.

Pregunta

Su simulador responde “no se detuvo en N pasos”. ¿Podría un programa mejor responder siempre “no se detiene nunca” cuando es el caso, para cualquier máquina y cualquier entrada?