Tema 5: Introducción a la Teoría de la Computabilidad

- Máquinas de Turing
- Implementación de tipos de datos en una MT.
- Problema de Parada
- Tesis de Church-Turing

OBJETIVOS:

- Utilización de la máquina de Turing como modelo computacional
- Comprensión de la existencia de límites intrínsecos a los procesos computacionales

Máquinas de Turing (**MT)**

MT es similar a un Autómata de Turing pero ahora NO HAY ESTADOS FINALES

- **Elementos**

Componentes físicos	Componentes lógicos
Unidad de Proceso	Conjunto de estados Q
Fuente de entrada	Alfabeto de entrada Σ
Memoria	Alfabeto de cinta Γ con Σ⊂Γ y con el símbolo blanco (□)

- **Elementos distinguidos** para la inicialización: $q_0 \in Q$ estado inicial
- **Ciclo-máquina:**

Consultas	Acciones
Estado actual Símbolo de cinta	Modificación de la cinta Cambio de estado *(escribir)* Mover cabeza lectora: *derecha (R) o izquierda (L)*

Definición formal:
$M = (Q, \Sigma, \Gamma, \delta, q_0)$ con $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L\}$

Funcionamiento de la Máquina de Turing

Configuración: $(u, q, v) \in \Gamma^* \times Q \times \Gamma^*$

Movimientos: (con $s,t,X \in \Gamma$ $u,w \in \Gamma^*$)

$(ut, p, sw) \vdash (utX, q, w)$ si y solo si $\delta(p, s) = (q, X, R)$

$(ut, p, sw) \vdash (u, q, tXw)$ si y solo si $\delta(p, s) = (q, X, L)$

Cómputo**:** Sucesión de movimientos desde la configuración inicial
Configuración inicial: $(\varepsilon, q_0, z_1 \square z_2 \square z_3 \square \ldots \square z_k)$ con $z_i \in \Sigma^*$ $(1 \le i \le k)$, $k \ge 1$.
Configuración final: La configuración obtenida al parar la máquina.

Resultado (del cómputo):

- Si (u, p, w) es la *configuración final* y $w = \mathbf{v}Xz$ con $\mathbf{v} \in \Sigma^*$, $X \in \Gamma - \Sigma$, $z \in \Gamma^*$ el *resultado del cómputo* es **v**.
  v es la palabra leída *desde la casilla señalada por la cabeza lectora* hacia la derecha, formada únicamente por *símbolos del alfabeto de entrada*.
- Si *no hay configuración final,* el resultado es indefinido (*diverge*)

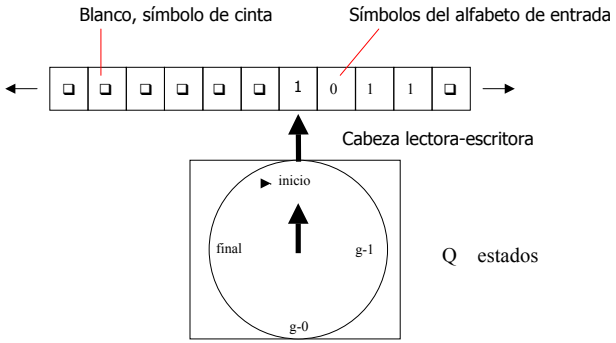
Ejemplo de máquina de Turing

Máquina que recibe un número en binario (cadena *no-vacía* de 0's y 1's) y devuelve el siguiente número binario (es decir, le suma 1).

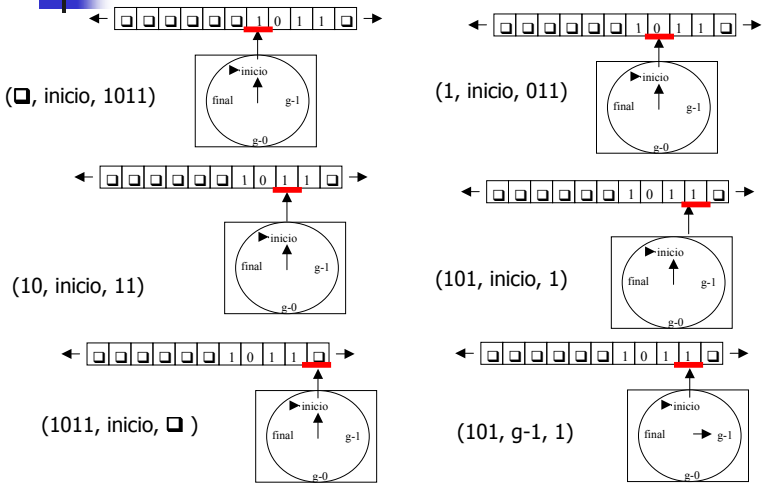
M = ({inicio, g-1, g-0, final}, {0, 1}, {0, 1, □}, **δ**, inicio)

δ	0	1	□
inicio	R	R	(g-1, □, L)
g-1	(g-0, 1, L)	(g-1, 0, L)	(final, 1, L)
g-0	L	L	(final, □, R)
final	---	---	R

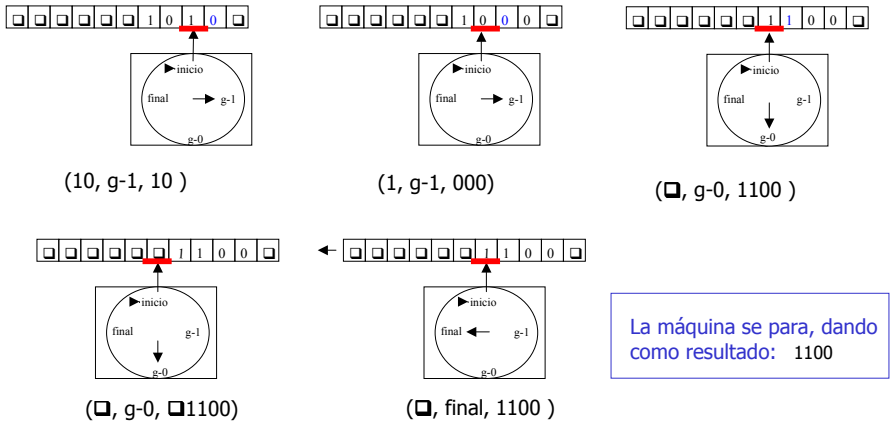
Ejemplo. Configuración inicial



Ejemplo de cómputo



Ejemplo de cómputo (sigue)



Funciones Computables

- Las máquinas de Turing pueden computar funciones (totales o parciales).
- Una función parcial está indefinida para algunos argumentos, para ellos la máquina de Turing debe ***diverger***.
- Para el resto de argumentos, la máquina debe dar como ***resultado de cómputo***: el resultado de la función sobre dichos argumentos.
- Una función f se dice que es Turing-computable si existe una máquina de Turing que la computa.

M computa la función $f : \Sigma^* \rightarrow \Sigma^*$ si:

$f(\mathbf{x}) = \mathbf{z} \Leftrightarrow (\varepsilon, q_0, \mathbf{x}) \vdash^* \text{conf. final con resultado } \mathbf{z}$

$f(\mathbf{x}) = \text{indefinida} \Leftrightarrow (\varepsilon, q_0, \mathbf{x}) \vdash^* \infty$ (M diverge, no para)

Ejemplo

Sea $\mathbf{f}: \Sigma^* \rightarrow \Sigma^*$ con $\Sigma=\{a, b\}$, definida:

$$\mathbf{f(u)} = \begin{cases} \boldsymbol{\varepsilon} & \text{si } \mathbf{u} \text{ contiene } \mathbf{a} \\ \uparrow & \text{en caso contrario} \end{cases}$$

Una MT para calcular **f** es $M = (\{q_0, q_1\}, \{a,b\}, \{a,b,\square\}, \delta, q_0)$ con δ definida:

δ	**a**	**b**	$\square$
q_0	(q_1, a, R)	R	R
q_1	R	R	---

Ejemplo: Máquina para invertir

- **Método**: escribir a la izquierda de la entrada **x**, de forma que la salida sea $\mathbf{x^R \# k}$ siendo $k = |\mathbf{x}|$ y $\Sigma = \{a, b\}$.
- La función $f(x) = x^R$ es Turing-computable.

δ	a	b	$\square$	#
q_0	$(q_a, \#, L)$	$(q_b, \#, L)$	$(q_\square, \square, L)$	R
q_a	L	L	$(q_\#, a, R)$	L
q_b	L	L	$(q_\#, b, R)$	L
$q_\#$	R	R	-	$(q_0, \#, R)$
$q_\square$	L	L	$(q_F, \square, R)$	L
q_F	---	---	---	---

Ejemplo: Máquina para concatenar x•y

- **Método**: eliminar el blanco que separa los dos argumentos **x** e **y**, moviendo los símbolos de **y** un lugar hacia la izquierda. $\Sigma = \{a, b\}$.
- La función f(x,y) = x.y es Turing-computable.

δ	a	b	$\square$
q_0	R	R	$(q_1, \square, R)$
q_1	$(q_a, \square, L)$	$(q_b, \square, L)$	$(q_{iz}, \square, L)$
q_a	-	-	(q_0, a, R)
q_b	-	-	(q_0, b, R)
q_{iz}	-	-	$(q_\square, \square, L)$
$q_\square$	L	L	$(q_F, \square, R)$
q_F	---	---	---

Diagramas de Máquinas de Turing

Diagramas: son grafos cuyos nodos son MTs y los arcos que los unen pueden venir etiquetados con símbolos de cinta.

Dadas dos MT: $M1 = (Q1, \Sigma, \Gamma, \delta_1, q_1)$ y $M2 = (Q2, \Sigma, \Gamma, \delta_2, q_2)$

- Concatenación: $M1 \rightarrow M2$

  Representa una nueva máquina **M** que comienza realizando los mismos movimientos que la máquina M1 y cuando M1 se para, comienza con los movimientos de M2 (con la cabeza lectora donde está y estado inicial q_2)

- Concatenación condicionada: $M1 \overset{a}{\rightarrow} M2$

  Representa una nueva máquina M que comienza realizando los mismos movimientos que la máquina M1 y cuando cuando M1 para, si la casilla señalada (por la cabeza lectora) contiene el símbolo **a**, comienza con los movimientos de M2.

Diagramas de Máquinas de Turing

- Iteración: 

Representa una nueva máquina M' que comienza realizando los mismos movimientos que la máquina M y cuando M para, si la casilla señalada (por la cabeza lectora) contiene el símbolo **a**, comienza de nuevo M, y así sucesivamente.

Máquinas de Turing básicas, sobre Σ= {a,b,c}

- **S** (START) *Está parada*

	a	b	c	□
q_0	---	---	---	---

- **R** (RIGHT) *Se mueve una posición a la derecha*

	a	b	c	□
q_0	(q_1,a,R)	(q_1,b,R)	(q_1,c,R)	$(q_1, □,R)$
q_1	---	---	---	---

- **L** (LEFT) *Se mueve una posición a la izquierda*

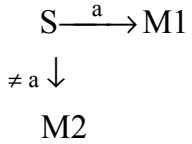
	a	b	c	□
q_0	(q_1,a,L)	(q_1,b,L)	(q_1,c,L)	$(q_1, □,L)$
q_1	---	---	---	---

- **Pa** (**a**$\in\Gamma$) (PRINT **a**) *Escribe el símbolo* **a** *y la cabeza-lectora apunta a dicho símbolo*

	a	b	c	□
q_0	(q_1,a,R)	(q_1,a,R)	(q_1,a,R)	(q_1, a,R)
q_1	(q_2,a,L)	(q_2,b,L)	(q_2,c,L)	$(q_2,□,L)$
q_2	---	---	---	---

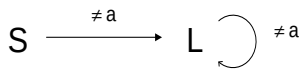
Máquinas de Turing que distinguen casos

Uso de la máquina **S** para el comienzo:

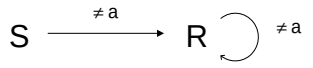


Máquinas de Turing que buscan el símbolo **a**

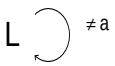
- **L_a^***: hacia la izquierda desde la casilla en la que está (incluida)



- **R_a^***: hacia la derecha desde la casilla en la que está (incluida)



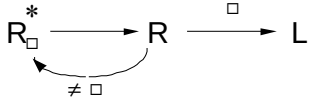
- **L_a^+**: desde la casilla izquierda de la casilla en la que está



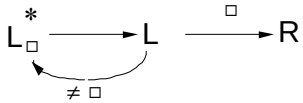
- **R_a^+**: desde la casilla derecha da la casilla en la que está

Máquinas de Turing que buscan los extremos

- **E** (End): busca el primer blanco a partir del cual hacia la **derecha** la cinta está vacía *(cuando hay dos blancos seguidos)*



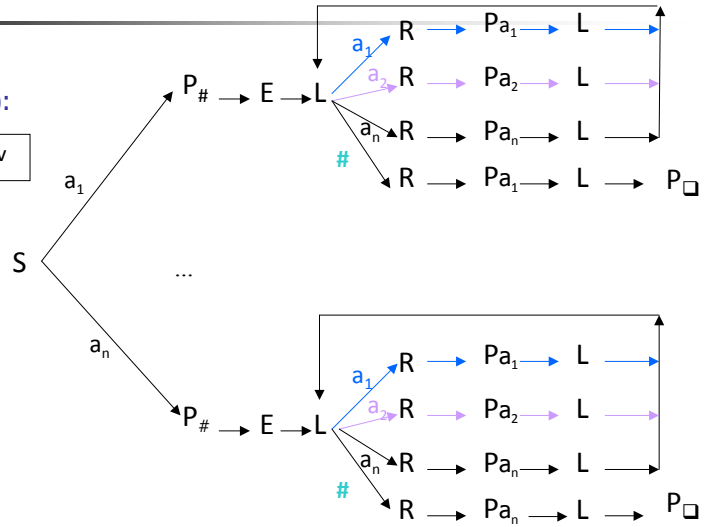
- **B** (Begin): busca el primer blanco a partir del cual hacia la **izquierda** la cinta está vacía



MT que abre un hueco a la derecha

H (Hole):

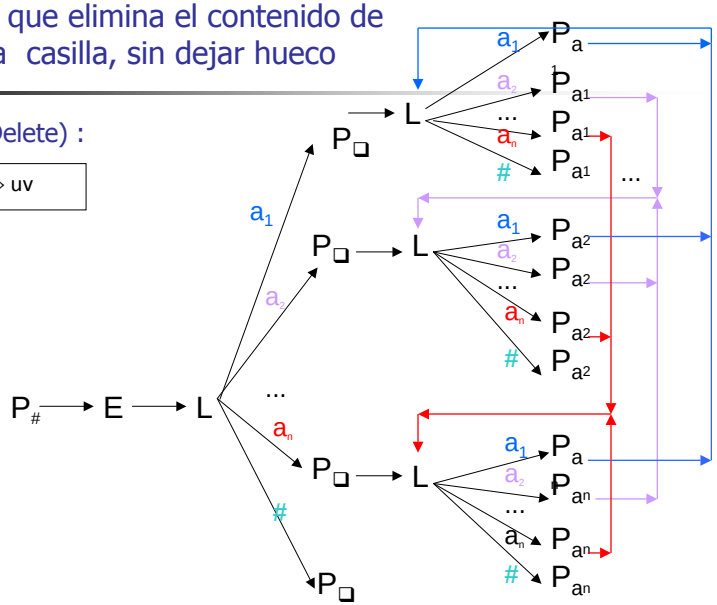
usv ⇒ u□sv



MT que elimina el contenido de una casilla, sin dejar hueco

D (Delete) :

usv ⇒ uv



Implementación de tipos de datos

Estudiaremos:

- La función ***siguiente*** que permite enumerar las palabras de un alfabeto
- Implementación de los **números naturales** y sus operaciones
- Implementación del **tipo de datos "pila"** y sus operaciones.

De manera similar se puede implementar cualquier tipo de datos.

La función ***siguiente*** (sig)

- Nos permite establecer un orden entre las palabras que se forman a partir de un alfabeto $\Sigma = \{\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_n}\}$
- Es Turing-computable
- Se define inductivamente:

$$sig(\varepsilon) = \mathbf{a_1}$$

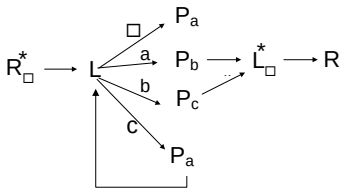
$$sig(w \bullet \mathbf{a_i}) = \begin{cases} w \bullet \mathbf{a_{i+1}} & i < n \\ sig(w) \bullet \mathbf{a_1} & i = n \end{cases}$$

La función ***siguiente*** sobre distintos Σ

	∑ = {a}	∑ = {a,b}	∑ = {a,b,c}	∑ = {0,1}
w_0	ε	ε	ε	ε
w_1	a	a	a	0
w_2	aa	b	b	1
w_3	aaa	aa	c	00
w_4	aaaa	ab	aa	01
w_5	aaaaa	ba	ab	10
w_6	aaaaaa	bb	ac	11
w_7	aaaaaaa	aaa	ba	000
w_8	aaaaaaaa	aab	bb	001
w_9	aaaaaaaaa	aba	bc	010
w_{10}	aaaaaaaaaa	abb	ca	011

La función ***siguiente*** es **computable**

Máquina de Turing, sobre Σ= {a,b,c}, que computa *siguiente* :



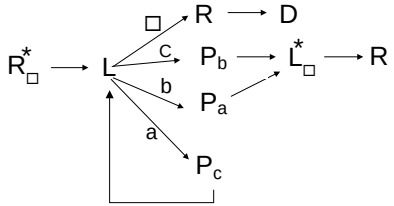
Implementación del tipo de datos **NAT**

N	∑ = {a}	∑ = {a,b}	∑ = {a,b,c}	∑ = {0,1}
0	ε	ε	ε	ε
1	a	a	a	0
2	aa	b	b	1
3	aaa	aa	c	00
4	aaaa	ab	aa	01
5	aaaaa	ba	ab	10
6	aaaaaa	bb	ac	11
7	aaaaaaa	aaa	ba	000
8	aaaaaaaa	aab	bb	001
9	aaaaaaaaa	aba	bc	010
10	aaaaaaaaaa	abb	ca	011

Implementando las operaciones de **NAT**

Máquinas de Turing, sobre $\Sigma = \{a,b,c\}$, que computan las operaciones

- *sucesor* es igual a *siguiente*
- *predecesor* o *anterior :*

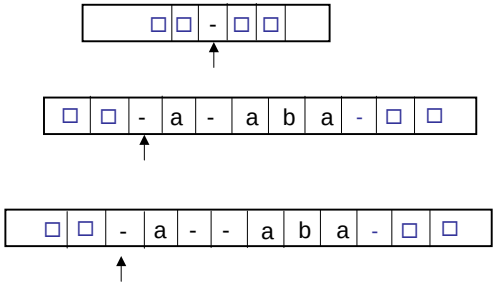


- *suma :*

Implementación del tipo de datos **PILA**

Supongamos las pilas (de elementos de un cierto tipo) representadas:

- pila vacía
- pila: a / aba
- pila: a / ε / aba



Implementando las operaciones de **PILA**

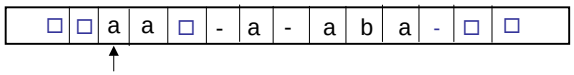
EJERCICIO Implementar las operaciones del tipo de datos PILA

apilar: $\Sigma^* \times$ Pila $\rightarrow$ Pila desapilar: Pila $\rightarrow$ Pila cima: Pila $\rightarrow \Sigma^*$

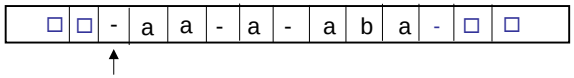
Ejemplo: apilar (aa, [a / aba]) = [aa / a / aba]

La máquina de Turing que implementa esta operación debe trasformar:

- Datos:



- Resultado:



Implementando las operaciones de **PILA**

Ejemplo:

desapilar ([aa / a / aba]) = [a / aba]

cima ([aa / a / aba]) = aa

Estas operaciones son parciales (indefinidas sobre la pila vacía) ➔ las máquinas correspondientes deben no parar sobre la pila vacía.

Codificación de máquinas de Turing

Cada máquina de Turing se puede codificar con una cadena binaria.

Dada M = ({q1, q2, ..., qk}, {0, 1}, Γ, δ, q1). Asignamos a cada elemento de la máquina un número entero positivo:

Estados: q1 → 1
q2 → 2
...
qk → k

Símbolos Γ = {X1, X2, ..., Xj}: X1 = 0 → 1
X2 = 1 → 2
X3 = □ → 3
...
Xj → j

Sentido del movimiento {S1, S2, S3}: S1 = L → 1
S2 = R → 2

Función de transición: $\delta(qi, Xj) = (qk, Xp, Sm) \rightarrow 0^i10^j10^k10^p10^m$

Código completo de la máquina de Turing M: $C_1 11 C_2 11 \ldots C_{n-1}\ 11 C_n$
donde cada C_i representa la codificación de una de las transiciones

Ejemplo de codificación de una MT

Sea **M**= ({q_1, q_2, q_3 }, {0, 1}, {0, 1, □}, δ, q_1). Con la función de transición δ:

$\delta(q_1, 1) = (q_3, 0, R)$
$\delta(q_3, 0) = (q_1, 1, R)$
$\delta(q_3, 1) = (q_2, 0, R)$
$\delta(q_3, □) = (q_3, 1, L)$

0 = X1, 1= X2, □ = X3
L = S1 , R = S2

Codificación de las transiciones:

$\delta(q_1,1) = (q_3,0,R)$	$\delta(q_1,X2) = (q_3,X1,S2)$	$0^110^210^310^110^2$ →	0100100010100
$\delta(q_3,0) = (q_1,1,R)$	$\delta(q_3,X1) = (q_1,X2,S2)$	$0^310^110^110^210^2$ →	0001010100100
$\delta(q_3,1) = (q_2,0,R)$	$\delta(q_3,X2) = (q_2,X1,S2)$	$0^310^210^210^110^2$ →	00010010010100
$\delta(q_3,□) = (q_3,1,L)$	$\delta(q_3,X3) = (q_3,X2,S1)$	$0^310^310^310^210^1$ →	0001000100010010

Una de las codificaciones posibles para M es:

0100100010100**11**000101010010**11**000100100101**11**0001000100010010

(Podemos variar el orden de las cuatro transiciones, por lo que tenemos 4!=24 codificaciones diferentes para esta máquina de Turing)

Enumeración de máquinas de Turing

- Cada máquina de Turing se puede codificar con una cadena binaria y a cada cadena binaria le corresponde un número (entero positivo).
  Por lo tanto:
  a cada máquina de Turing le corresponde un número
- A cada número le corresponde una cadena binaria y a cada cadena binaria le corresponde una máquina de Turing (si asociamos a las cadenas que no son códigos adecuados la máquina **S**).
  Por lo tanto:
  a cada número le asociamos una máquina de Turing
- Entonces, se tiene la correspondencia:

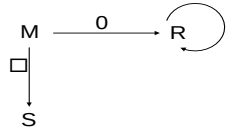
número i ← → máquina $\mathbf{M_i}$

Problema de parada

La función de parada no es Turing-computable.

$$\textbf{Halt}(x) = \begin{cases} 0 & M_x \text{ con entrada x para} \\ \varepsilon & M_x \text{ con entrada x no para} \end{cases}$$

Demostración:
Por reducción al absurdo. Supongamos que existe una máquina **M** que computa la función Halt. Construimos con ella una nueva máquina de la siguiente forma:



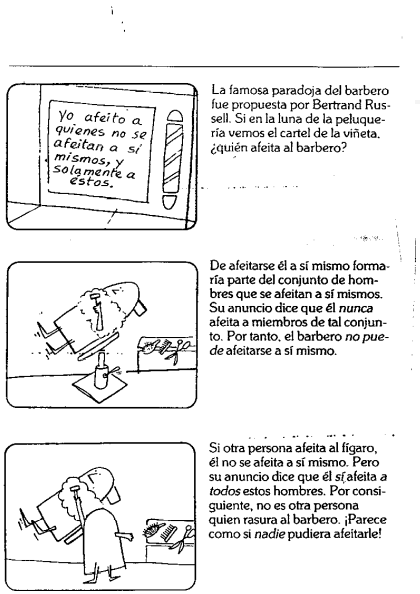
La máquina construida tendrá un código **e**.
Es, por tanto, la máquina $\mathbf{M_e}$

¿Qué ocurre si la máquina $\mathbf{M_e}$ recibe como entrada su propio código?

Si Halt(e) = 0 ⇒ **M** devuelve 0 ⇒ $\mathbf{M_e}$ no para ⇒ Halt(e) = ε ↯
Si Halt(e) = ε ⇒ **M** devuelve ε ⇒ $\mathbf{M_e}$ para ⇒ Halt(e) = 0 ↯ **¡Absurdo!**

Conclusión: No existe una máquina **M** que compute la función Halt.

La paradoja del barbero



La famosa paradoja del barbero fue propuesta por Bertrand Russell. Si en la luna de la peluquería vemos el cartel de la viñeta, ¿quién afeita al barbero?

De afeitarse él a sí mismo formaría parte del conjunto de hombres que se afeitan a sí mismos. Su anuncio dice que él *nunca* afeita a miembros de tal conjunto. Por tanto, el barbero *no puede* afeitarse a sí mismo.

Si otra persona afeita al fígaro, él no se afeita a sí mismo. Pero su anuncio dice que él *sí* afeita *a todos* estos hombres. Por consiguiente, no es otra persona quien rasura al barbero. ¡Parece como si *nadie* pudiera afeitarle!

Tesis de Church-Turing

La Tesis de Church-Turing:

Todo sistema de computación es A LO SUMO TAN POTENTE como las máquinas de Turing

- Esta afirmación vale para modelos **pasados**, **presentes** y **futuros**.
- No puede ser demostrada, pero podría ser refutada.
- Su corolario es muy importante:

Si una función no es Turing-computable, NO EXISTE UNA SOLUCIÓN (O ALGORITMO) para la misma en ningún sistema de computación.

Otras formalizaciones equivalentes

λ-cálculo	A. Church	1935
Funciones recursivas	S. Kleene	1935
Máquinas de Turing	A. Turing	1936
Máquinas de Post	E. Post	1936
Sistemas de Thue	A. Thue	1912
Cadenas de Markov	A. Markov	1947
Máquinas de registros	N. Cutland	1962
Programas lógicos	J. W. Lloyd	1970
Programas-while	A.J. Kfouri, R.N. Moll, M.A. Arbib	1982

Principios de la Informática Teórica

- Las operaciones primitivas imprescindibles en un sistema computacional son muy pocas. La mayor parte de las utilidades suministradas por los lenguajes de programación son superfluas.
- Todos los sistemas computacionales son equivalentes y sirven para solucionar exactamente los mismos problemas, aunque pueden diferenciarse en la facilidad que presentan para resolverlos.
- Hay problemas que ningún programa puede resolver. La mayor parte de ellos están relacionados con la propia Informática y disciplinas afines.
- El comportamiento de los programas es esencialmente imprevisible. En general, los problemas dependientes de su ejecución no tienen solución.
- Los problemas computables tienen asociados costes intrínsecos que ningún programa que los resuelva puede soslayar.
- Las leyes de la Informática no son susceptibles de cambiar mediante avances tecnológicos en el futuro.