Lenguajes de máquinas de Turing

Horst H. von Brand
vonbrand@inf.utfsm.cl

Departamento de Informática
Universidad Técnica Federico Santa María

Contenido

Numerar palabras

Es más fácil/natural razonar con (colecciones de) números naturales, veremos maneras de numerar los objetos de interés: palabras y autómatas. Ya discutimos una manera de representar una máquina de Turing como palabra sobre $\Sigma = \{0, 1\}$, veamos ahora cómo traducir entre palabras y números.

Orden lexicográfico

Sea un alfabeto $\Sigma = \{a_1, a_2, \ldots, a_n\}$ (en algún orden acordado). Podemos ordenar las palabras de Σ^* en *orden lexicográfico* (como el diccionario):

- La primera es ε
- Enseguida las palabras de largo 1, en el orden $a_1, a_2, \ldots, a_n$
- Después largo 2, $a_1a_1, a_1a_2, \ldots, a_1a_n, a_2a_1, \ldots, a_2a_n, \ldots, a_na_n$
- Luego palabras de largo 3, $a_1a_1a_1, \ldots, a_na_na_n$.
- ...

Orden lexicográfico

Como sansanos astutos que son, ven que esto corresponde a decir que $\sigma = a_{x_1} a_{x_2} \cdots a_{x_m}$ está en la posición:

$$\sum_{1 \leq k \leq m} x_k \cdot n^{m-k} + 1$$

donde x_k es el número del símbolo del caso en el orden acordado. De acá es fácil derivar un algoritmo que da la palabra dado su orden en la lista (realmente es una biyección, la *numeración biyectiva*).

Numeración biyectiva

El algoritmo para calcular el número correspondiente a la palabra $\sigma = a_{x_1} a_{x_2} \ldots a_{x_m}$ es calcular un polinomio en n:

function string2int($a_{x_1} a_{x_2} \ldots a_{x_m}$)
    $r \leftarrow x_m$
    for $k = m - 1$ **downto** 1 **do**
        $r \leftarrow r \cdot n + a_k$
    end
    return $r + 1$
end

El caso del alfabeto $\{0, 1\}$ es más simple: agregue un 1 al comienzo, lea el resultado como un número escrito en binario.

Numeración biyectiva

El algoritmo para calcular la palabra correspondiente a un natural es algo más complicado:

function number2string(n)
    $\sigma \leftarrow \varepsilon$
    repeat
        $q \leftarrow \lceil n/b \rceil - 1$
        $k \leftarrow n - q \cdot b$
        $\sigma \leftarrow a_k \cdot \sigma$
    until $n = 0$
    return σ
end

Lenguajes que ninguna máquina de Turing reconoce

Con lo anterior vemos que las palabras (y en consecuencia las máquinas de Turing) forman un conjunto numerable. Por el teorema de Cantor, hay más subconjuntos de los números naturales que números naturales, o sea, hay más lenguajes que máquinas de Turing. Hay lenguajes que ninguna máquina de Turing acepta.

El lenguaje diagonal

Daremos un ejemplo concreto de lenguaje que ninguna máquina de Turing acepta.
Por la discusión anterior, podemos hablar de la palabra número i sobre $\Sigma = \{0, 1\}$, que llamaremos σ_i. Por nuestra codificación de máquinas de Turing podemos igualmente hablar de la máquina de Turing número i, que bautizaremos M_i.

Definición

El *lenguaje diagonal* es:

$$L_d = \{\sigma_i : \sigma_i \notin \mathscr{L}(M_i)\}$$

L_d no es aceptado por ninguna máquina de Turing

Teorema

Ninguna máquina de Turing acepta L_d.

Demostración.

Por contradicción. Supongamos que M_k acepta L_d. Por la definición de L_d, es $\sigma_k \in \mathscr{L}(M_k) = L_d$ si y solo si $\sigma_k \notin L_d$. Esto es absurdo. □

Relación con «¡Hola, mundo!»

Si se compara la demostración anterior de que no hay programa que determina si un programa cualquiera escribe «¡Hola, mundo!» con la demostración del teorema, el razonamiento es esencialmente el mismo.

Tipos de lenguajes

De lo tratado anteriormente, vemos que hay lenguajes que son aceptados por máquinas de Turing, y lenguajes que ninguna máquina de Turing acepta. Son de particular interés los lenguajes aceptados por máquinas de Turing que siempre se detienen (garantizan entregar una respuesta Sí/No en tiempo finito; basta de armarse de paciencia, habrá respuesta).

Tipos de lenguajes

Definición

Un lenguaje que es aceptado por una máquina de Turing se llama *computablemente enumerable*, un lenguaje aceptado por una máquina de Turing que siempre se detiene se llama *computable* o *decidible*.

Es claro que los lenguajes decidibles son computacionalmente enumerables.

Nomenclatura tradicional (lentamente en retirada) es llamarlos *recursivamente enumerables* y *recursivos*, respectivamente.

Computacionalmente enumerable se abrevia ce (tradicionalmente re).

Razón de los nombres

Lo de *computablemente enumerable* se refiere a que dada una máquina de Turing M podemos confeccionar una lista (enumeración) de las palabras de $\mathscr{L}(M)$ como sigue: Podemos generar sistemáticamente pares (i,k), para no omitir ninguno en orden de $i+k$ creciente y dentro de los pares que suman n por i creciente. Para cada par (i,k), ejecutamos M sobre σ_i por k pasos. Si M acepta σ_i, añadimos σ_i a la cinta de salida.

Razón de los nombres

Podemos incluso poner la restricción que la lista no tenga duplicados: antes de agregarla a la lista, basta revisar si ya está.

Cuidado, no podemos poner como condición por ejemplo que las palabras estén ordenadas. Si entrega las palabras en orden, basta esperar a que aparezca la palabra (pertenece al lenguaje) o aparece una palabra posterior (no pertenece). Sería un lenguaje decidible.

Ejemplos de cada tipo de lenguaje

Bonitas definiciones. . . ¿Existen realmente lenguajes de las distintas categorías?

Ejemplos de cada tipo de lenguaje

Computable: Los lenguajes regulares son computables (decidibles).

Computablemente enumerable: Todos los aceptados por máquinas de Turing.

No computablemente enumerable: Dimos el ejemplo L_d.

Ejemplos de cada tipo de lenguaje

Claro que la pregunta realmente era si hay lenguajes computacionalmente enumerables que no son computables. (Todos los lenguajes computables son computablemente enumerables, por definición.)

Un ejemplo es el lenguaje aceptado por la máquina de Turing universal U: si fuera decidible, podríamos decidir L_d (para determinar si $\sigma_i \in L_d$, ejecute U sobre $\langle M_i \rangle \sigma_i$, responda «sí» si U no acepta), pero L_d ni siquiera es computablemente enumerable.

Clasificación de $\overline{L}_d$

Teorema

$\overline{L}_d$ es computablemente enumerable.

Demostración.

Primero, $\overline{L}_d = \{\sigma_i : \sigma_i \in \mathscr{L}(M_i)\}$. Podemos enumerar $\overline{L}_d$ vía generar todos los pares (i,k) sistemáticamente para ejecutar M_i sobre σ_i por k pasos. Si M_i acepta σ_i en k pasos, listamos σ_i. □

Relación entre L y $\overline{L}$

Una pregunta más o menos obvia es por la relación entre la clasificación de L y $\overline{L}$. Exploraremos las posibilidades.

L computable

Si L es computable, claramente también lo es $\overline{L}$.

L y $\overline{L}$ computablemente enumerables

Teorema

Si L y $\overline{L}$ son computablemente enumerables, ambos son computables.

Demostración.

Dadas máquinas de Turing deterministas M con $L = \mathscr{L}(M)$ y $\overline{M}$ con $\overline{L} = \mathscr{L}(\overline{M})$, corremos M y $\overline{M}$ en paralelo mediante una máquina de Turing de dos cintas (o dando pasos alternadamente en cada cinta), la primera que acepte nos dice si $\sigma \in L$ o $\sigma \in \overline{L}$, con lo que tenemos que una máquina de Turing que *siempre* se detiene que acepta L (o $\overline{L}$). □

L computablemente enumerable, no computable

Corolario

Si L es computablemente enumerable, no computable, $\overline{L}$ no es computablemente enumerable.

Demostración.

Directo del teorema anterior. □

L y $\overline{L}$ no computablemente enumerables

Última categoría posible.

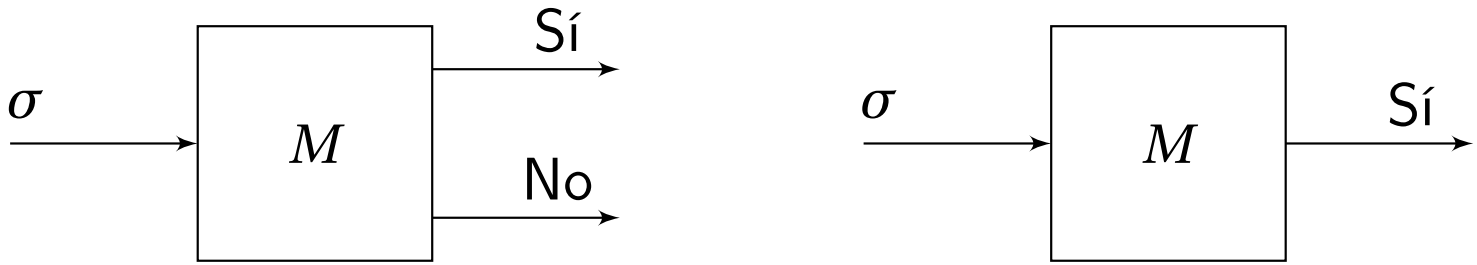
Estos son lenguajes *realmente* inalcanzables. Daremos un ejemplo luego.

Demostración por cajitas

Muchas veces un diagrama es la manera más clara de explicar algo.
La idea es representar cada máquina de Turing por una cajita, indicando cómo reconoce a través de sus salidas.

Demostración por cajitas

Esquematizamos una máquina de Turing que siempre se detiene como la figura 1a. A una máquina de Turing que podría no detenerse la forzamos a entrar en un ciclo infinito si se detiene sin aceptar. El esquema es 1b.



(a) computable

(b) computablemente enumerable

Figura: Esquemas de máquinas de Turing.

Demostración por cajitas

Como ejemplo, repetiremos una demostración previa.
Suponemos dadas cajas que representan algunas máquinas conocidas, como la máquina universal U. Usamos construcciones como duplicar la corriente de entrada, que es copiar la entrada a una nueva cinta; tener dos cajas en paralelo, lo que representa correr las máquinas en pasos alternados sobre sus respectivas cintas de trabajo; «inventar» palabras usando no-determinismo; ignorar la entrada es trabajar en una nueva cinta, nunca considerando la cinta original. Cuando usamos una máquina de Turing empotrada en otra, es que copiamos la máquina interior (renombrando estados y ajustando las transiciones del estado final) de manera de lograr el efecto de «subrutina».

L y $\overline{L}$ computablemente enumerables

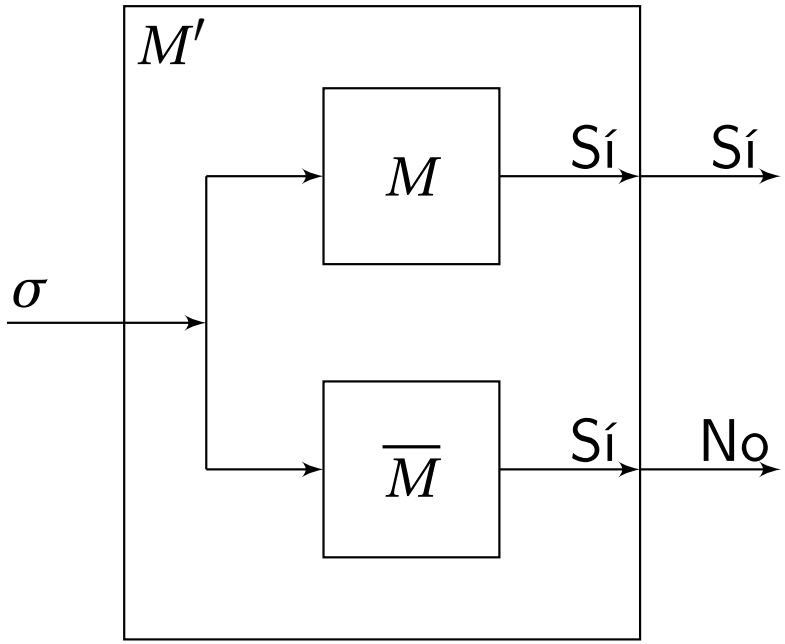
Teorema

Si L y $\overline{L}$ son computablemente enumerables, ambos son computables.

L y $\overline{L}$ computablemente enumerables

Demostración.

Dadas máquinas de Turing M y $\overline{M}$ tales que $L = \mathscr{L}(M)$ y $\overline{L} = \mathscr{L}(\overline{M})$, la máquina de Turing de la figura decide L.



Algunas propiedades de clausura

Usando cajitas podemos demostrar que los lenguajes computables:

- Son cerrados respecto a las operaciones regulares (unión, concatenación, estrella de Kleene).
- Son cerrados respecto de operaciones entre conjuntos (unión, intersección, complemento).

¿Cuáles de las anteriores *no* valen para lenguajes computacionalmente enumerables?

Problemas

Para nuestros efectos, un *problema* no es más que un lenguaje sobre algún alfabeto Σ. Lo que pide es determinar si $\sigma \in \Sigma^*$ (una *instancia del problema*) pertenece o no al lenguaje.

Problemas, reducciones, . . .

La demostración de lo siguiente es solo aplicar las definiciones:

Teorema

Sean $\mathscr{P}_1, \mathscr{P}_2$ problemas (subconjuntos de Σ^*), con $\mathscr{P}_1 \leq \mathscr{P}_2$. Entonces:

- *Si $\mathscr{P}_2$ es decidible, $\mathscr{P}_1$ es decidible.*
- *Si $\mathscr{P}_1$ no es decidible, $\mathscr{P}_2$ no es decidible.*
- *Si $\mathscr{P}_2$ es computacionalmente enumerable, $\mathscr{P}_1$ es computacionalmente enumerable.*
- *Si $\mathscr{P}_1$ no es computacionalmente enumerable, $\mathscr{P}_2$ no es computacionalmente enumerable.*

Un par de lenguajes útiles

Al demostrar problemas no decidibles (no computacionalmente enumerables) es útil contar con lenguajes simples con las características de interés.

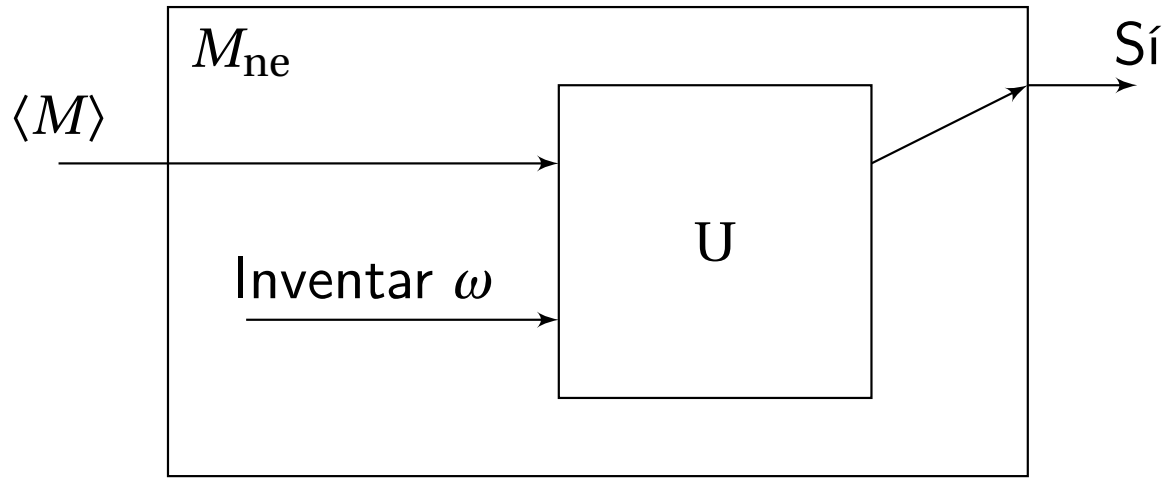
$$L_{\mathrm{e}} = \{\langle M\rangle : \mathscr{L}(M) = \varnothing\} \qquad \textit{empty}$$
$$L_{\mathrm{ne}} = \{\langle M\rangle : \mathscr{L}(M) \neq \varnothing\} \qquad \textit{not empty}$$

Para referencia futura, son complementos.

L_{ne} es computacionalmente enumerable

Demostración.

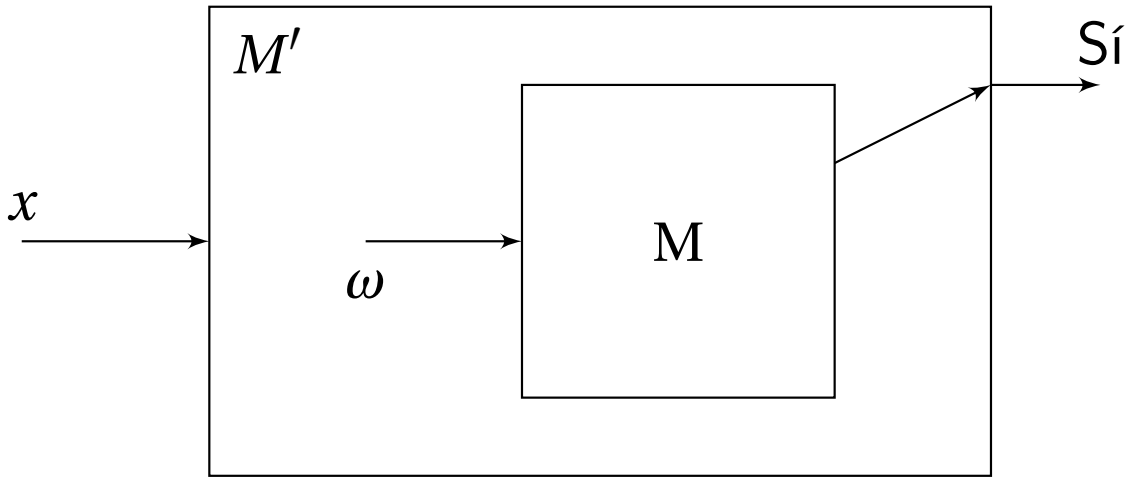
Adaptamos la máquina de Turing universal: no-deterministamente inventamos una palabra ω y simulamos M sobre ella.



L_{ne} no es decidible

Demostración.

Sabemos que L_U, el lenguaje de U, no es decidible. Demostramos $L_U \leq L_{\text{ne}}$. Dados M, ω construimos la máquina de Turing M' del diagrama (descarta su entrada y simula M sobre ω). Decidir si M' acepta $\varnothing$ es decidir si $\omega \in \mathscr{L}(M)$, cosa que sabemos imposible.



L_{e} no es computacionalmente enumerable

Demostración.

Antes demostramos que $\overline{L}_{\mathrm{e}} = L_{\mathrm{ne}}$ es computacionalmente enumerable, no decidible. □

No computacionalmente enumerable con su complemento

Discutiremos un lenguaje que no es computacionalmente enumerable, y su complemento tampoco. Con esto tenemos ejemplos concretos de todos los casos.

No computacionalmente enumerable con su complemento

Definimos $L_\infty = \{\langle M \rangle : \mathscr{L}(M) \text{ es infinito}\}$.

Teorema

L_∞ no es computablemente enumerable.

No computacionalmente enumerable con su complemento

Demostración.

reduciendo $\overline{L}_{\mathrm{U}} \leq L_{\infty}$. Dada una máquina de Turing M y una palabra ω, construimos M' que acepta un lenguaje infinito si y solo si M no acepta ω. La máquina de Turing M' toma un natural n como entrada y simula M sobre ω por a lo más n pasos. Si M no ha aceptado en n pasos, M' acepta. O sea, $\mathscr{L}(M')$ es infinito (es todo $\mathbb{N}$) si y solo si $(\langle M \rangle, \omega) \notin \mathscr{L}(M')$. Concluimos L_{∞} no es computacionalmente enumerable. □

No computacionalmente enumerable con su complemento

Teorema

$\overline{L}_\infty$ no es computablemente enumerable.

No computacionalmente enumerable con su complemento

Demostración.

Reducimos $\overline{L}_{\mathrm{U}} \leq \overline{L}_{\infty}$. Dada una máquina de Turing M y una palabra ω, construimos M'' que acepta un lenguaje finito si y solo si M no acepta ω. La construcción de M'' es similar a la de M' de la parte anterior, solo que acepta si M ha aceptado a los n pasos. O sea, $\mathscr{L}(M'')$ es finito (es vacío) si y solo si $(\langle M \rangle, \omega) \notin \mathscr{L}(M')$.
Concluimos que $\overline{L}_{\infty}$ no es computacionalmente enumerable. □

El teorema de Rice

Una *propiedad* (de un lenguaje) es simplemente el conjunto de lenguajes con esa propiedad.

Definición

Una propiedad de los lenguajes se dice *no trivial* si hay lenguajes que tienen la propiedad y lenguajes que no la tienen.

Representamos un lenguaje computacionalmente enumerable por una máquina de Turing que lo acepta.

El teorema de Rice

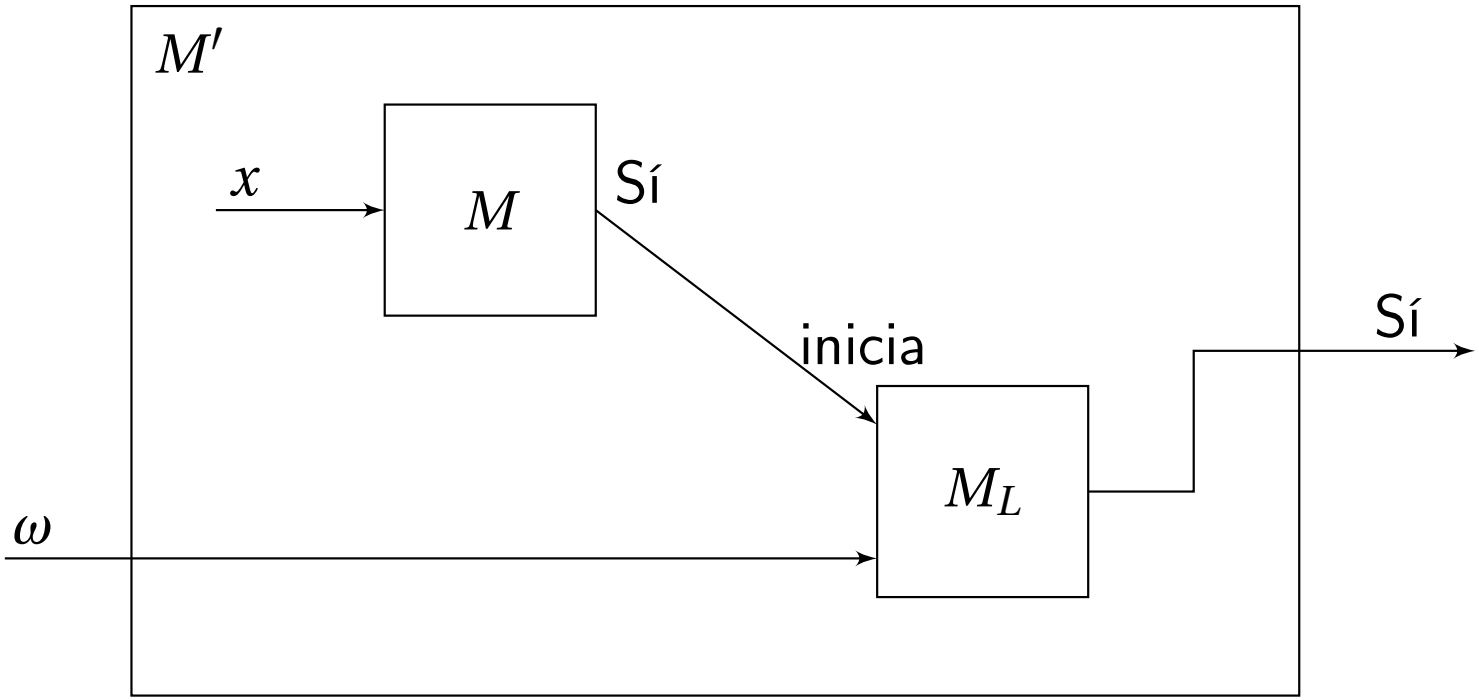
Teorema (Rice)

Toda propiedad no trivial de los lenguajes computacionalmente enumerables no es decidible.

El teorema de Rice

Demostración.

Sea la propiedad $\mathscr{P}$. Primero consideremos $\varnothing \notin \mathscr{P}$, sea $L \in \mathscr{P}$ aceptado por la máquina de Turing M_L. Dados M, x construimos M' como la figura. Decidir si $\mathscr{L}(M') \in \mathscr{P}$ es decidir si $x \in \mathscr{L}(M)$.



El teorema de Rice

Falta el caso $\varnothing \in \mathscr{P}$. Basta considerar la propiedad no trivial $\overline{\mathscr{P}}$, que sabemos $\varnothing \notin \overline{\mathscr{P}}$, y aplicar la parte anterior. $\square$

Observación

El teorema de Rice habla de *el lenguaje aceptado por* M, no de M ni directamente de su comportamiento.
Por ejemplo, «$\mathscr{L}(M)$ es regular» es una propiedad no trivial, no es decidible. Ver si M es un DFA es muy simple (solo avanza).

Resumen

- Numeramos palabras y máquinas de Turing.
- Exhibimos un lenguaje que ninguna máquina de Turing acepta.
- Definimos computable (decidible) y computablemente enumerable. Dimos ejemplos de lenguajes de los distintos tipos.
- Exploramos clasificación de L y $\overline{L}$.
- Definimos diagramas para máquinas de Turing y construcciones que las involucran.
- Clasificamos L_{e} (no computablemente enumerable) y su complemento L_{ne} (computablemente enumerable, no decidible).
- Clasificamos L_{∞} (no computablemente enumerable) y su complemento $\overline{L}_{\infty}$ (no computablemente enumerable).
- Demostramos el teorema de Rice.