Autómatas y Lenguajes Formales
Nota 14. Lenguajes recursivos y recursivamente enumerables. Reducciones entre lenguajes*

Noé Salomón Hernández S.

1. Algoritmos *vs* procedimientos

Algoritmos Se asocian a máquinas de Turing (MT*'s*) totales, es decir, MTs que siempre se detienen en un tiempo finito bajo cualquier entrada y cualquier trayectoria de ejecución, ya sea para aceptar o rechazar. Este tipo de MTs corresponde a nuestra noción informal de un "algoritmo", una secuencia bien definida de pasos que siempre termina y produce una respuesta.

La clase de lenguajes aceptados por los algoritmos se llaman **recursivos** o **decidibles**.

Los lenguajes que no son recursivos, o no tienen algoritmos, se llaman **indecidibles**.

Procedimientos Se asocian a máquinas de Turing arbitrarias, las cuales pueden no detenerse bajo una entrada $w \notin L$.

La clase de lenguajes aceptados por los procedimientos se llaman **recursivamente enumerables (r.e.)** o **reconocibles por máquina de Turing**.

Las clases de lenguajes que hemos estudiado y la relación entre ellas se representan en el diagrama de la Figura 1.

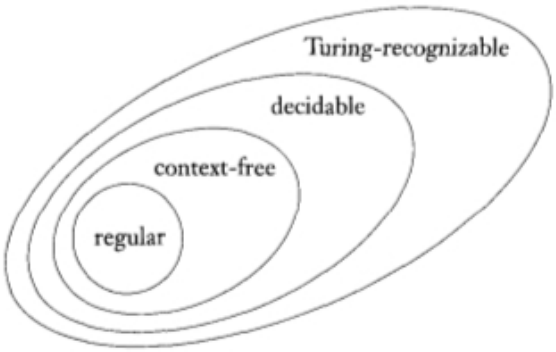

Figura 1: La relación entre las clases de lenguajes.

*Esta nota se basa en los libros de J. Martin, *Introduction to Languages and the Theory of Computation*; de M. Sipser, *Introduction to the Theory of Computation*; y de D.C. Kozen, *Automata and Computability*. También en las notas del profesor Rajeev Motwani.

1.1. Propiedades de cerradura

Teorema 1.1 *Si L es recursivo, entonces $\overline{L}$ es recursivo.*

Teorema 1.2 *Si ambos L y $\overline{L}$ son recursivamente enumerables, entonces L y $\overline{L}$ son ambos recursivos.*

De lo anterior se sigue que, para cualquier lenguaje L se cumple una de las siguientes afirmaciones:

- L y $\overline{L}$ son ambos recursivos, o
- al menos uno de los dos, L o $\overline{L}$, no es recursivamente enumerable (r.e.)

Esto se resume en la Figura 2.

	$\overline{L}$ es recursivo	$\overline{L}$ es r.e. pero no recursivo	$\overline{L}$ no es r.e.
L es recursivo	✓	X	X
L es r.e. pero no es recursivo	X	X	✓
L no es r.e.	X	✓	✓

Figura 2: Relación de los lenguajes recursivos y r.e. con sus complementos.

Teorema 1.3 *Si L_1 y L_2 son recursivos, entonces $L_1 \cup L_2$ es recursivo.*

Teorema 1.4 *Si L_1 y L_2 son r.e., entonces $L_1 \cup L_2$ es r.e.*

2. Codificación de una máquina de Turing

Asumimos que hay un conjunto infinito $\mathcal{S} = \{a_1, a_2, a_3, \ldots\}$ de símbolos, donde $a_1 = \sqcup$, tal que el alfabeto de la cinta de cualquier máquina de Turing M es un subconjunto de $\mathcal{S}$.

La idea de codificación que usaremos es simplemente representar una máquina de Turing como un conjunto de transiciones, las cuales se representan como cadenas de $0s$ y $1s$. En cada transición

$$\delta(p, a) = (q, b, D)$$

se contemplan cinco componentes $p, q \in Q$; $a, b \in \Sigma$; y $D \in \{L, R\}$, cada uno de los cuales tiene un número asignado; por ejemplo, tomemos p y le asignamos $n(p)$. A dicho número se le asocia una cadena de unos, en nuestro ejemplo asociamos la cadena $1^{n(p)}$. Así, la quintupla se representa por la cadena que contiene a estos cinco bloques de 1s, cada uno seguido por un 0.

Definición 2.1 Una función de codificación

Si $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ es una MT, con $F = \{q_f\}$, y w es una cadena, definimos las cadenas binarias que codifican a M y a w, $e(M)$ y $e(w)$, como sigue.

Primero asignamos números a cada estado, símbolo de la cinta, y dirección de la cabeza lectora de M. Cada símbolo de la cinta, incluyendo a $\sqcup$, es un elemento de $a_i \in \mathcal{S}$, y se le asigna el número

$n(a_i) = i$. Al estado final q_f y al estado inicial q_0 se les asignan los números $n(q_f) = 1$ y $n(q_0) = 2$. A los otros estados $q \in Q$ se les asignan números distintos, empezando en 3. No requerimos que tales números sean consecutivos, y el orden no es importante. Finalmente, las dos direcciones R y L se representan con los números $n(R) = 1$ y $n(L) = 2$.

Para cada transición m en M de la forma $\delta(p, \sigma) = (q, \tau, D)$ la codificación queda como

$$e(m) = 1^{n(p)}01^{n(\sigma)}01^{n(q)}01^{n(\tau)}01^{n(D)}0$$

Enumeramos las transiciones de M en algún orden, digamos $m_1, m_2, \ldots, m_k$, y definimos

$$e(M) = e(m_1)0e(m_2)0 \ldots 0e(m_k)0$$

Si $w = w_1w_2 \ldots w_j$ es una cadena de entrada, donde cada $w_i \in \mathcal{S}$ para toda $1 \leq i \leq j$, su codificación queda como

$$e(w) = 01^{n(w_1)}01^{n(w_2)}0 \ldots 01^{n(w_j)}0$$

3. Máquina de Turing universal

Definición 3.1 Máquina de Turing Universal (MTU)

Una máquina de Turing universal es una MT M_u que funciona como sigue. Suponemos que recibe cadenas de entrada de la forma $e(M)e(w)$, donde M es una máquina de Turing arbitraria, w es una cadena sobre el alfabeto de entrada de M, y e es una función de codificación como la que se muestra en la Definición 2.1. No hay confusión acerca de donde termina $e(M)$ y donde comienza $e(w)$, esto ocurre en la primera presencia de 000; los dos primeros 0s marcan el término de $e(M)$ y el tercero es el comienzo de $e(w)$. Cada transición en $e(M)$ está separada por dos 0s, el primero es con el que se termina la codificación de una transición m y el segundo es el cero después de las $e(m_i)$, como lo indica $e(M)$. La descripción de como opera M_u se dará en la demostración del Teorema 5.2.

Es de notar que el cómputo realizado por M_u sobre la cadena de entrada $e(M)e(w)$ safisface estas propiedades:

1. M_u acepta la cadena $e(M)e(w)$ si y solo si M acepta w.
2. Si M acepta w e imprime como salida a y en la cinta, entonces M_u imprime como salida a $e(y)$.

A las codificaciones $e(M)$, $e(w)$ y $e(M)e(w)$ también se les denota como $\langle M \rangle$, $\langle w \rangle$ y $\langle M, w \rangle$, respectivamente.

Observemos que una cadena binaria que es una codificación válida corresponde a una única máquina de Turing. Pero si la cadena binaria no es una codificación válida, supondremos que codifica la máquina particular $M_\varnothing$ que tiene un estado y ninguna transición, de modo que $L(M_\varnothing) = \varnothing$. Así cada máquina de Turing M puede ser codificada como una cadena binaria $\langle M \rangle$ tal que:

- Cada MT tiene al menos una codificación.
- Cada número binario codifica una única MT.

3.1. Enumerando cadenas binarias

A continuación definimos un orden específico de todas las cadenas binaria finitas. Primeramente, las ordenamos de manera incremental a partir de su longitud, y a las cadenas binarias de la misma longitud las ordenamos lexicográficamente, i.e., siguiendo el orden del diccionario. Por lo que tenemos:

$$\varepsilon,\ 0,\ 1,\ 00,\ 01,\ 10,\ 11,\ 000,\ 001,\ 010,\ 011,\ 100,\ 101,\ 110,\ 111, \ldots$$

Sea w_i la i-ésima cadena en esta enumeración, con $w_1 = \varepsilon$.

Definimos a M_i como la MT codificada por w_i. De este modo obtenemos un ordenamiento de las MT*'s* en el cual cada MT figura al menos una vez, pero posiblemente muchas veces.

4. El lenguaje de la diagonal

Tomaremos como el alfabeto de entrada a $\{0, 1\}$ para las máquinas de Turing de esta sección en adelante, por lo que no es necesario aplicar la codificación $e(w)$ a las cadenas de entrada w, ya que podemos trabajar con w directamente. Consideremos la enumeración de cadenas binarias en la Sección 3.1 para construir la tabla infinita Π. Para toda $i, j \in \mathbb{N}$ se tiene

$$\Pi(i, j) = \begin{cases} 1 & w_i \in L(M_j) \\ 0 & w_i \notin L(M_j) \end{cases}$$

La tabla Π podría verse como la que aparece en la Figura 3. Recuerde que $w_1, w_2, w_3, w_4, w_5 \ldots$ corresponde a la enumeración de cadenas binarias discutida arriba, es decir, $w_1, w_2, w_3, w_4, w_5 \ldots$ corresponde a $\varepsilon, 0, 1, 00, 01, \ldots$, respectivamente. De igual manera, las codificaciones de máquinas de Turing $\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle M_5\rangle, \ldots$ corresponden a la enumeración de cadenas binarias $\varepsilon, 0, 1, 00, 01, \ldots$, respectivamente.

	w_1	w_2	w_3	w_4	w_5	$\ldots$
$\langle M_1\rangle$	0	1	1	0	0	$\ldots$
$\langle M_2\rangle$	1	1	0	0	1	$\ldots$
$\langle M_3\rangle$	0	0	0	1	0	$\ldots$
$\langle M_4\rangle$	1	0	1	0	1	$\ldots$
$\langle M_5\rangle$	1	1	1	0	1	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Figura 3: La tabla Π *que toma las codificaciones de máquinas de Turing* M_i *y las cadenas de entrada* w_i*.*

Notemos que la tabla Π puede ser postulada porque hemos mostrado un ordenamiento/enumeración de w_i y $\langle M_i\rangle$. Además, cada renglón de Π es un vector característico de $L(M_i)$ mostrando las cadenas que le pertenecen.

Definición 4.1 El lenguaje de la diagonal

$$L_d = \{w_i \in \{0, 1\}^* \mid \Pi(i, i) = 0\} = \{w_i \in \{0, 1\}^* \mid w_i \notin L(M_i)\}$$

L_d está definido al tomar la diagonal de Π, reemplazando 0 por 1, y 1 por 0. Así obtenemos un vector característico para el lenguaje L_d.

Nos surge la pregunta, ¿es L_d r.e.? ¿es L_d recursivo?

Teorema 4.2 *L_d no es r.e.*

Demostración:

Supongamos que L_d es r.e. y tiene un MT. Por lo que existe una $k \in \mathbb{N}$ tal que $L(M_k) = L_d$. ¿Pertenece w_k a L_d?

Caso 1. Sí, w_k pertenece a L_d. Tenemos que:

$$\begin{aligned} w_k \in L_d &\Rightarrow w_k \in L(M_k) \\ &\Rightarrow \Pi(k,k) = 1 \\ &\Rightarrow w_k \notin L_d \quad \textbf{¡Contradicción!} \end{aligned}$$

Caso 2. No, w_k no pertenece a L_d. Tenemos que:

$$\begin{aligned} w_k \notin L_d &\Rightarrow w_k \notin L(M_k) \\ &\Rightarrow \Pi(k,k) = 0 \\ &\Rightarrow w_k \in L_d \quad \textbf{¡Contradicción!} \end{aligned}$$

Al suponer que L_d es r.e. llegamos a una contradicción. Por lo tanto, L_d **no** es r.e.

⊣

La intuición para que L_d no sea r.e. es que L_d se definió para diferir de cada M_i en al menos la cadena w_i. Esto implica que ninguna de las M_i*'s* puede tener a L_d como el lenguaje que reconoce. Pero como todas las MT*'s* M están codificadas como $\langle M_i \rangle$ para algún i, concluimos que no hay MT que reconozca a L_d. La existencia de L_d fuera de los lenguaje r.e. se debe a que el conjunto infinito de cadenas binarias es numerable, mientras que el conjunto infinito de todos los lenguajes binarios no es numerable. Así, hay más lenguajes binarios que máquinas de Turing que los reconocen.

5. L_u es r.e. pero no es recursivo

Definición 5.1 El lenguaje universal (L_u)
El lenguaje universal L_u es

$$L_u = \{\langle M, w \rangle \mid w \in L(M)\}$$

Teorema 5.2 *El lenguage L_u es r.e.*

Demostración:

El lenguaje L_u es reconocido por la máquina de Turing universal (MTU) M_u (ver Definición 3.1). La MTU M_u tiene cuatro cintas.

Cinta 1. Es una cinta de sólo lectura. Contiene a $\langle M, w\rangle$.

Cinta 2. Simula a la cinta de M. Inicialmente copiamos w a esta cinta.

Cinta 3. Contiene al estado actual q_i de M.

Cinta 4. Cinta auxiliar que puede guardar otros estados, símbolos o direcciones.

La ejecución de M_u simula paso a paso el comportamiento de M sobre w. Inicialmente copiamos w a la cinta 2, y q_0 a la cinta 3. A cada paso usamos el contenido de las cintas 2 y 3 para saber el estado actual y el símbolo que se está leyendo, con lo que se determina la siguiente transición. Para simular este movimiento, M_u tiene que buscar en la cinta 1 por la quintupla cuyos primeros dos bloques correspondan al estado y símbolo en cuestión, los últimos tres bloques nos indican como llevar a cabo el movimiento, de modo que modificamos las demás cintas como corresponda. M_u acepta si se llega a un estado final de M.

$\dashv$

5.1. El lenguaje L_u no es recursivo

Hemos visto que L_u es r.e. ya que $L_u = L(M_u)$. Ahora veremos que L_u no es recursivo. Para esto necesitaremos de la noción de **reducción entre lenguajes**.

Definición 5.3 L_1 se reduce a L_2, denotado $L_1 \prec L_2$, si existe una función f, llamada **reducción**, tal que:

a) La aplicación de f sobre cualquier cadena w es calculada por una máquina de Turing total, la cual se detiene con $f(w)$ como el contenido final de la cinta. Recuerde que una máquina de Turing total es una MT que siempre se detiene.

b) f es tal que para toda cadena w

$$w \in L_1 \Leftrightarrow f(w) \in L_2$$

Por lo que f mapea todas las cadenas en L_1 a un subconjunto de cadenas en L_2. Además, todas las cadenas en $\overline{L_1}$ se mapean a un subconjunto de cadenas en $\overline{L_2}$.

Teorema 5.4 *Si $L_1 \prec L_2$ y L_2 es recursivo, entonces L_1 es recursivo.*

Demostración:

Se discutirá en clase.

Corolario 5.5 *Si $L_1 \prec L_2$ y L_1 no es recursivo, entonces L_2 no es recursivo.*

Los dos resultados anteriores pueden aplicarse también a lenguajes r.e. como se indica a continuación.

Teorema 5.6 *Si $L_1 \prec L_2$ y L_2 es r.e., entonces L_1 es r.e.*

Corolario 5.7 *Si $L_1 \prec L_2$ y L_1 no es r.e., entonces L_2 no es r.e.*

Los Corolarios 5.5 y 5.7 pueden ser usados para demostrar que un lenguaje no es recursivo, o bien, no es r.e., respectivamente.

Teorema 5.8 *El lenguaje L_u no es recursivo.*

Demostración:

Como sabemos que L_d no es r.e., por la tabla en la Figura 2 concluimos que $\overline{L_d}$ no es recursivo. Así, para concluir que L_u no es recursivo mostraremos $\overline{L_d} \prec L_u$ y aplicaremos el Corolario 5.5.

Recordemos que $L_d = \{w_i \in \{0,1\}^* \mid w_i \notin L(M_i)\}$, luego $\overline{L_d} = \{w_i \in \{0,1\}^* \mid w_i \in L(M_i)\}$. También recuerde que $w_i = \langle M_i \rangle$. Así, $\overline{L_d}$ es el conjunto de las cadenas binarias que son aceptadas por la MT M que codifican.

Considere como reducción f aquella que toma a una cadena binaria w y produce la cadena $f(w) = \langle w, w \rangle$. Claramente existe una MT total M_f que convierte w en $\langle w, w \rangle$. Necesitamos mostrar que $w \in \overline{L_d} \Leftrightarrow f(w) \in L_u$. Observe que w es una cadena binaria, así que tiene asignado un indice i que indica el lugar que ocupa w en la enumeración de cadenas binarias (ver Sección 3.1), así $w = w_i$. Tenemos que

$$\begin{aligned} w_i \in \overline{L_d} &\Leftrightarrow w_i \in L(M_i) \\ &\Leftrightarrow \langle M_i, w_i \rangle \in L_u \\ &\Leftrightarrow \langle w_i, w_i \rangle \in L_u \\ &\Leftrightarrow f(w_i) \in L_u \end{aligned}$$

Con esto $\overline{L_d} \prec L_u$. Además, $\overline{L_d}$ no es recursivo. Por el Corolario 5.5, L_u no es recursivo.

6. El problema del paro o de la detención

El problema de la detención consiste en determinar si una MT M se detiene bajo una cadena de entrada w. Visto como lenguaje se especifica como sigue:

$$L_H = \{\langle M, w \rangle \,|\, M \text{ se detiene al procesar } w \text{ como entrada}\}$$

Teorema 6.1 *El lenguaje L_H no es recursivo.*

Demostración. Vamos a reducir el lenguaje L_u a L_H para luego aplicar el Corolario 5.5. Como L_u no es recursivo, concluimos que L_H tampoco es recursivo.

Sabemos que $L_u = \{\langle M, w \rangle \mid w \in L(M)\}$, así que a partir de $\langle M, w \rangle$ deseamos construir $f(\langle M, w \rangle) = \langle N, w \rangle$ tal que

$$\langle M, w \rangle \in L_u \Leftrightarrow \langle N, w \rangle \in L_H.$$

La descripción de la MT N depende de M y w. En particular, N tiene la descripción de M de manera pre-cargada en su control de estados finito.

La manera en que funciona N sobre w consiste en los siguientes pasos:

a) cede el control a M, quien se ejecuta sobre w;

b) si M acepta a w, entonces N se detiene procesando w;

c) si M se detiene no aceptando a w, entonces N entra en un ciclo infinito.

El ciclo que se genera en el inciso c) puede consistir en dos estados nuevos, $\dot{q}$ y $\ddot{q}$, de modo que de uno se llega al otro bajo cualquier símbolo de la cinta, y N llega a $\dot{q}$ una vez que M se detiene en cualquier estado no final.

Si M acepta a w, la máquina N que construimos se detiene procesando w. Si M se detiene rechazando a w, la máquina N entra en un ciclo infinito como lo indica c), por lo que no se detiene al procesar w. Si M no se detiene sobre w, la simulación en el inciso a) nunca se detiene, por lo que N tampoco se detiene al procesar w. Así

$$\begin{array}{lcl} M \text{ acepta } w & \Rightarrow & N \text{ se detiene procesando } w, \\ M \text{ rechaza } w \text{ deteniéndose} & \Rightarrow & N \text{ no se detiene procesando } w, \\ M \text{ rechaza } w \text{ al no detenerse} & \Rightarrow & N \text{ no se detiene procesando } w. \end{array}$$

Es decir,

$$\begin{array}{lcl} M \text{ acepta } w & \Rightarrow & N \text{ se detiene procesando } w, \\ M \text{ no acepta } w & \Rightarrow & N \text{ no se detiene procesando } w. \end{array}$$

Luego, M acepta w syss N se detiene procesando w. Por lo tanto, $\langle M, w\rangle \in L_u \Leftrightarrow \langle N, w\rangle \in L_H$.

Observemos que la máquina de Turing total para $f(\langle M, w\rangle)$, que produce una descripción de $\langle N, w\rangle$ a partir de M y w, no tiene que ejecutar el programa descrito por los incisos a) – c). Sólo produce la descripción de una máquina N que lleva a cabo tal ejecución. El cómputo que hace f es sencillo, únicamente toma una descripción de una MT M y una cadena w, y los enchufa dentro de una descripción general de una máquina que realice lo estipulado por a) – c). En otras palabras, $f(\langle M, w\rangle)$ genera una cadena binaria que corresponde a $\langle N, w\rangle$, es fácil ver que escribir una cadena binaria finita lo puede llevar a cabo una MT total.

Así tenemos que, $L_u \prec L_H$. Dado que L_u no es recursivo, y por el Corolario 5.5, L_H no es recursivo.

⊣

Tenemos la siguiente lista de lenguajes y las clases a las que pertenecen.

- L_d **no es r.e.** Ver Teorema 4.2.
- $\overline{L_d}$ **es r.e. pero no es recursivo.** Ejercicio de la Tarea 8.
- L_u **es r.e. pero no es recursivo.** Es r.e. porque la máquina de Turing universal M_u lo reconoce. No es recursivo porque se reduce de $\overline{L_d}$.
- $\overline{L_u}$ **no es r.e.** Por el punto anterior y por la tabla de la Figura 2.
- L_H **es r.e. pero no es recursivo.** Es r.e. porque la máquina de Turing universal M_u puede ser usada para reconocer L_H. No es recursivo porque se reduce de L_u.
- $\overline{L_H}$ **no es r.e.** Por el punto anterior y por la tabla de la Figura 2.

Ejemplo 6.2 Por medio de una reducción demuestre que el lenguaje $L_{\Re} = \{N \mid N \text{ es una MT y } L(N) \text{ es un lenguaje regular}\}$ no es recursivo.

Ejemplo 6.3 Demuestre que el lenguaje $L_{TOT} = \{\langle M \rangle \mid M \text{ se detiene con todas las entradas}\}$ no es recursivo.

Demostración. Deseamos reducir L_H a L_{TOT}, es decir, buscamos hallar f, función que es ejecutada por una MT total, para la cual

$$\langle M, x \rangle \in L_H \Leftrightarrow f(\langle M, x \rangle) \in L_{TOT}.$$

En otras palabras, buscamos $M' = f(\langle M, x \rangle)$ que cumpla

$$M \text{ se detiene al procesar } x \Leftrightarrow M' \text{ se detiene con todas las entradas.}$$

Dada $\langle M, x \rangle$, M' con la entrada y funciona de la siguiente manera:

1. Borra la entrada y.
2. Escribe x en la cinta de M'.
3. M' simula a M con la entrada x.
4. Si M se detiene con x, entonces M' se detiene con la entrada y.

Notemos que M' guarda en su control de estados finito a x y a la descripción de M, es decir, en las transiciones de M' se tiene "alambrado" como escribir x en la cinta y también los movimientos para la ejecución de M.

Ahora, si M se detiene al procesar x, entonces M' se detiene con y, esto es así para toda entrada y. Si M no se detiene con x, entonces nunca se alcanza el último paso y M' se queda simulando a M con x, de manera que M' tampoco se detiene con y, lo cual pasa para toda entrada y. Por consiguiente,

$$\begin{aligned} M \text{ se detiene al procesar } x &\Rightarrow M' \text{ se detiene con toda entrada } y, \\ M \text{ no se detiene al procesar } x &\Rightarrow M' \text{ no se detiene con ninguna entrada } y \\ &\Rightarrow M' \text{ no se detiene con todas las entradas.} \end{aligned}$$

Las dos implicaciones anteriores satisfacen que

$$M \text{ se detiene al procesar } x \Leftrightarrow M' \text{ se detiene con todas las entradas.}$$

Es decir, se satisface

$$\langle M, x \rangle \in L_H \Leftrightarrow f(\langle M, x \rangle) \in L_{TOT}.$$

Luego $L_H \prec L_{TOT}$, como L_H no es recursivo, tampoco lo es L_{TOT} siguiendo el Corolario 5.5.

Finalmente, tenemos que f es una función que indica la manera de construir la MT M' pero no ejecuta el programa ni simula M en ningún momento, simplemente da una descripción de M' que en última instancia es una codificación para dicha MT, que como vimos corresponde a una cadena binaria finita. Lo cual puede llevarse a cabo por una máquina de Turing total.

7. El teorema de Rice

Fijamos un alfabeto de entrada Σ. Una propiedad de los lenguajes recursivamente enumerables es una función

$$P : \{\text{lenguajes r.e. sobre } \Sigma^*\} \to \{V, F\}.$$

Por ejemplo, la propiedad de un lenguaje de ser vacío se especifica como la función:

$$P(A) = \begin{cases} V & \text{si } A = \varnothing, \\ F & \text{si } A \neq \varnothing. \end{cases}$$

Si deseamos preguntarnos si una propiedad P es decidible, el lenguaje tiene que representarse de forma apropiada para tomarse como entrada de una MT. Suponemos que los lenguajes r.e. son representados a través de las MT que los aceptan. Debemos estar conscientes que la propiedad es una propiedad de los lenguajes, no de las máquinas de Turing; así que debe ser verdadera o falsa independientemente de la MT en particular que hayamos escogido para representar al conjunto.

Aquí hay algunas propiedades de los lenguajes r.e.: $L(M)$ es finito; $L(M)$ es regular; $L(M)$ es LLC; $10111101 \in L(M)$; $L(M) = \Sigma^*$. Cada una de estas propiedades en una propiedad del lenguaje aceptado por una máquina de Turing.

Aquí hay algunos ejemplos de propiedades de máquinas de Turing que no son propiedades de los lenguajes r.e.: M tiene al menos 1000 estados; M se detiene en todas las entradas; M rechaza a 10111101. Estas no son propiedades de los lenguajes r.e. porque en cada caso podemos dar dos MTs que acepten el mismo lenguaje; una de las cuales satisfaga la propiedad y la otra no.

Una propiedad es no trivial si no es universalmente verdadera ni universalmente falsa, es decir, debe de haber al menos un lenguaje r.e. que satisface la propiedad y al menos uno que no.

Teorema 7.1 *Toda propiedad no trivial de los lenguajes r.e. es indecidible.*

Demostración. Se encuentra en el libro de D.C. Kozen, *Automata and Computability*, Springer-Verlag, Inc., New York, NY, 1997.