2.2.

2.2.1. Máquina de Turing universal

Fijemos un alfabeto A. Podemos considerar que todas las máquinas de Turing sobre A son deterministas, tienen como alfabeto de la cinta $\{0, 1, \hbar\}$ y alfabeto de entrada $\{0, 1\}$ (al fin y al cabo, todos los alfabetos pueden codificarse en binario). También pueden modificarse para que sólo tengan un estado final, que no hay problema en llamar q_2. Así que, toda máquina de Turing puede reformularse para tener

$$Q = \{q_1, q_2, \ldots\}$$
$$\Gamma = \{0, 1, \hbar\}$$
$$F = \{q_2\}$$

de forma que una máquina de Turing sólo se diferencia de otra en el número de estados y la función de transición. Pero la función de transición no es más que una sucesión de elementos $(q, e) \rightarrow (q', e', m)$ donde $m \in \{\leftarrow, \rightarrow\}$.

Supongamos ahora que renombramos 0 como X_1, 1 como X_2 y $\hbar$ como X_3, $\leftarrow$ como D_1 y $\rightarrow$ como D_2. Entonces, una máquina de Turing es una sucesión de elementos (transiciones) $(q_i, X_j), (q_k, X_l, D_m)$, que también podemos expresar como $1^i01^j01^k01^l01^m$. Podemos utilizar 00 para separar los elementos, de forma que una máquina de Turing concreta estaría perfectamente determinada por la "ristra"

$$1^{i_1}01^{j_1}01^{k_1}01^{l_1}01^{m_1}001^{i_2}01^{j_2}01^{k_2}01^{l_2}01^{m_2}00\ldots1^{i_p}01^{j_p}01^{k_p}01^{l_p}01^{m_p}$$

siendo p el número de transiciones (casillas no vacías de su tabla de transición).

Dada una máquina de Turing M, llamaremos $< M >$ a su codificación de esta manera. También, dada una cadena de entrada para M, $W \in \Sigma^*$, puede codificarse para dar $< w >$.

Una máquina de Turing *universal* (MTU) es una máquina con tres cintas. En la primera de ellas se pone la codificación de una máquina de Turing $< M >$. En la segunda, la codificación de M, $< w >$. En la tercera, la codificación de q_1 (o sea, 1).

MTU concatena el contenido de la tercera cinta, un cero, y lo que haya en la segunda hasta el primer cero (el primer símbolo de w), o sea $< q_1 > 0 < w_1 >$ y busca en la primera cinta esta subcadena al principio o después de 00 (busca la transición aplicable), y encuentra $< q_1 > 0 < w_1 > 0 < q' > 0 < e' > 0 < m >$, así que pone en la tercera cinta $< q' >$ en lugar de $< q_1 >$ y $< e' >$ en lugar de $< w_1 >$ (para lo cual tendrá posiblemente que desplazar el resto de símbolos). Es decir, hace en la segunda cinta lo que haría $< M >$. Continúa de la misma manera. Si en algún momento no encuentra la subcadena que busca, se para, que es lo que habría hecho M en el mismo caso.

Dicho de otro modo: imita el comportamiento de M sobre su entrada w. Es decir, ejecuta el programa M sobre su entrada w. O sea, que MTU es un computador de programa almacenado.

2.2.2. Numerabilidad de las M.T.

De la codificación vista en el apartado anterior se puede deducir, que a cada máquina de Turing podría asociársele un número: el que represente en binario esa "ristra" de ceros y unos $< M >$ (en realidad varios, según el orden en que se pongan las transiciones; supongamos que tomamos el menor de los posibles). Así que, el conjunto de máquinas de Turing, es numerable. [1]

Algunas máquinas de Turing serán generadoras. Y todo lenguaje recursivamente numerable tendrá alguna máquina de Turing que la genere (incluso varias). Por lo tanto hay a lo sumo $\aleph_0$ lenguajes recursivamente numerables. Pero $|\mathcal{P}(\Sigma^*)|$ **no** es numerable; por lo tanto hay lenguajes que no son recursivamente numerables (y muchísimos).

El problema es que es difícil describir uno de ellos, precisamente porque al no ser recursivamente numerables, no hay *algoritmo* que los pueda describir. Sin embargo, identificaremos uno de ellos.

[1] No todos los números enteros positivos corresponderán a máquinas de Turing

2.2.3. Ejemplo de lenguaje no recursivamente numerable

Pongamos $\{0,1\}^*$ en su orden canónico:

$$\begin{aligned} w_1 &= \lambda \\ w_2 &= 0 \\ w_3 &= 1 \\ w_4 &= 10 \\ w_5 &= 11 \\ w_6 &= 100 \\ &\dots \end{aligned}$$

y "numeremos las máquinas de Turing $M_1, M_2, M_3, \dots$[2].

Dada una cadena w_i y una máquina M_j, ocurrirá que, o bien $M_j(w_i)$ se para aceptando o no. Definamos $T[i, j] = 1$ si ocurre lo primero y $T[i, j] = 0$ en el segundo. Es decir, T es una tabla

	M_1	M_2	...	M_j	...
w_1	0	0	...	1	...
w_2	0	1	...	0	...
...	...	...	...	...	...
w_i	1	1	...	1	...
...	...	...	...	...	...

donde en $T[i, j]$ pone 1 si M_j se para aceptando w_i y 0 en caso contrario.[3]

Ahora sea

$$L_U = \{w_i / T[i, i] = 0\}$$

es decir, las cadenas que corresponden a un punto de la diagonal en la que hay un 0. Resulta que L_U es el conjunto de cadenas w_i tales que la máquina con el mismo número M_i *no* reconoce (no responde 'Si').

Pues bien, L_U **no es recursivamente numerable**, porque no puede haber ninguna máquina que lo reconozca.

Si la hubiera, estaría en la lista, y se llamaría M_p.

El problema está en qué hace M_p con w_p:

- Si la acepta, entonces $w_p \in L_U$. Pero también entonces $T[p, p] = 1$, de donde $w_p \notin L_U$. Esto es absurdo.
- Si no la acepta, entonces $w_p \notin L_U$, luego $T[p, p] \neq 0$, así que es 1, por lo que M_p no la acepta. esto también es absurdo.

Conclusión: no puede haber tal máquina aceptadora de L_U

2.2.4. Existe de un lenguaje recursivamente numerable no recursivo

Por ejemplo, $L_D = \overline{L_U}$. Como acabamos de ver que su complementario (L_U) no es recursivamente numerable, menos será recursivo, y por lo tanto, L_D tampoco (el complementario de un recursivo también lo es). Pero sí es recursivamente numerable: obsérvese que

$$L_D = \{w_i / T[i, i] = 1\}$$

Se puede construir una M.T. que lo reconozca, de las que no se garantiza que se pare siempre:

[2] no es que realmente *vayamos a hacerlo*, es que existen tales numeraciones y se podrían calcular algorítmicamente.

[3] No se pretende *construir* la tabla, dado que no se podría, ni disponiendo de tiempo.

leer (x)
localizar la posición de x en el orden : $x = w_i$
cargar MTU con el programa M_i
ejecutar $MTU(M_i, w_i)$
si se para aceptando **entonces**
  escribir 'Si'
fin si

Evidentemente, si $x \in L_D$, M_i se para aceptando $w_i = x$, luego este "casi algoritmo" también; y si no, se pare o no, nunca dirá 'Si'. Por lo tanto, este proceso de cálculo responde afirmativamente ante las cadenas de L_D y sólo ante ellas. Así que L_D es recursivamente numerable.

2.2.5. Máquinas de Turing y jerarquía de Chomsky

Una gramática de tipo 0 es un algoritmo generador. Luego todo lenguaje de tipo 0 es recursivamente numerable.

Recíprocamente, si se tiene una M.T. M que reconozca L (se pare o no siempre), se puede construir una gramática que genere L, imitando el proceso reconocedor en orden inverso. Reescribiendo las configuraciones $(q, \alpha\underline{a}\beta)$ como $\alpha qa\beta$), la sucesión de configuraciones del reconocimiento se transforma en una sucesión de cadenas que parten q_1x y terminan en una configuración $\alpha q_f\beta$. Un estudio más detallado permite la construcción de una gramática que produce, como formas sentenciales, estas configuraciones.

Así que los lenguajes recursivamente numerables resultan ser, exactamente, los de tipo 0.

Por otra parte, todo lenguaje de tipo 1 es recursivo, ya que, como se vió anteriormente, cualquier gramática no contractiva permite definir un algoritmo de reconocimiento de los que siempre se paran.

Sin embargo, hay lenguajes de tipo 1 que no son recursivos. Para probarlo se puede usar un argumento de construcción de

Así que los lenguajes recursivamente numerables resultan ser, exactamente, los de tipo 0.

Por otra parte, todo lenguaje de tipo 1 es recursivo, ya que, como se vió anteriormente, cualquier gramática no contractiva permite definir un algoritmo de reconocimiento de los que siempre se paran.

Sin embargo, hay lenguajes de tipo 1 que no son recursivos. Para probarlo se puede usar un argumento de construcción de un conjunto "diagonal" como el empleado en

Así que los lenguajes recursivamente numerables resultan ser, exactamente, los de tipo 0.

Por otra parte, todo lenguaje de tipo 1 es recursivo, ya que, como se vió anteriormente, cualquier gramática no contractiva permite definir un algoritmo de reconocimiento de los que siempre se paran.

Sin embargo, hay lenguajes de tipo 1 que no son recursivos. Para probarlo se puede usar un argumento de construcción de un conjunto "diagonal" como el empleado en (2.2.3): se numeran todas las gramáticas de tipo 1, y se construye $L_N = \{w_i / w_i \notin L(G_i)\}$ L_N no puede ser de tipo 1, porque ninguna de las numeradas puede generarlo (volveríamos a encontrar un absurdo al plantear la generación de la cadena del mismo número). Pero sí es recursivo, porque, dada una cadena w_i, basta construir la gramática G_i y comprobar si $w_i \in L(G_i)$ para saber si w_i está o no en L_N.

En resumen

Proposición 2.2.5.1 *Todo lenguaje de tipo 0 es recursivamente numerable.*

Proposición 2.2.5.2 *Todo lenguaje recursivamente numerable es de tipo 0.*

Proposición 2.2.5.3 *Todo lenguaje de tipo 1 es recursivo.*

Proposición 2.2.5.4 *Existen lenguajes recursivos que no son de tipo 1.*

2.2.6. Autómatas linealmente acotados

La máquina correspondiente al nivel de los lenguajes de tipo 1 es el *autómata linealmente acotado*: una máquina de Turing en la que la cabeza no se desplaza fuera de los límites entre los que inicialmente se sitúa la cadena de entrada.

3. Funciones recursivas

Las máquinas de Turing pueden también verse como calculadoras de funciones de $\Sigma^* \rightarrow \Gamma^*$. Ambos alfabetos permitirán codificar por ejemplo a los números naturales, o incluso a tuplas de naturales, de forma que se puede plantear qué funciones podrían calcularse con una máquina de Turing y cuáles no.

Se llamará *función parcialmente computable* a una función definida sobre una parte de N^r, con resultado en N que podría calcularse con una máquina de Turing. O de otro modo: se pone a funcionar la máquina de Turing sobre una entrada de N^r. Si se para, habrá calculado el resultado, y la función estará definida en ese punto. Si no, la función no estará definida en ese punto.

Se llamará *función computable* a una función definida en todo N^r, con resultado en N que sea parcialmente computable. Es decir, para la que exixta una máquina de Turing que la calcule, y que se pare siempre (siempre dé un valor).

El estudio de las funciones recursivas es precisamente el de las funciones computables.