Temas finales de
Teoría de Autómatas y Lenguajes Formales
II
Curso 2002-2003

M. Luisa González Díaz
Departamento de Informática
Universidad de Valladolid

2. Máquinas de Turing

2.1.

2.1.1. Definición, representación y simulación

Una máquina de Turing es un "dispositivo" como lo eran los autómatas finitos o los autómatas a pila, con más capacidades que éstos. Dispone también de un número finito de estados, uno de ellos inicial, y algunos de ellos finales. Dispone también de una *cinta*, que es una sucesión "doblemente infinita" de "celdas", en cada una de las cuales hay un símbolo. La cinta está inicialmente "en blanco" salvo en una porción finita, en la que está almacenada la entrada. La máquina de Turing puede leer y escribir símbolos en la cinta, y moverse a lo largo de ella en ambos sentidos. Para ello dispone de una *cabeza* de lectura-escritura. Su operación viene determinada por su *función de transición*.

$$M = \{\Gamma, \Sigma, \hbar, Q, q_0, f, F)$$

donde

Γ es el alfabeto de la cinta

Σ es el alfabeto de entrada, con $\Sigma \subseteq \Gamma$

$\hbar \in \Gamma - \Sigma$ es el símbolo *blanco*

Q es un conjunto finito de *estados*

$q_0 \in Q$ es el *estado inicial*

$F \subseteq Q$ es el conjunto de estados *finales*

f es la *función de transición*:

$$f : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\leftarrow, \rightarrow\})$$

La función de transición determina, dado un estado q y un símbolo e que la cabeza lee en la cinta, lo que puede hacer la máquina. Si $f(q, e) = \{(q', e', m) \ldots\}$, una de las posibilidades será (q', e', m), que quiere decir que la máquina escribirá en la cinta el símbolo e' (en la celda donde estaba e, es decir, sustituyendo e por e'), moverá la cabeza según indique m (una celda a la izquierda o a la derecha) y pasará al estado q'.

Inicialmente, la máquina comienza en el estado inicial q_0, en la cinta hay una cadena de Σ^* (y el resto en blanco, $\hbar$), y la cabeza se encuentra sobre el primer símbolo de la cadena de entrada (o sobre un blanco si la cadena es vacía).

La configuración en un momento dado vendrá dada por

- el estado en el que se encuentra
- el contenido completo de la cinta. Como la parte de la cinta que no está en blanco es una cadena finita, bastará con indicar este fragmento, sabiendo que a su izquierda y a su derecha habrá infinitas celdas en blanco.
- la posición de la cabeza de lectura-escritura

Ejemplo:

$$\Gamma = \{|, \bullet, \hbar\} \quad \Sigma = \{|\} \qquad q_0 = p \quad F = \{s\} \qquad f:$$

$MT1$	$\|$	$\bullet$	$\hbar$
p	$q\ \bullet\ \rightarrow$	$p\ \bullet\ \leftarrow$	$r\ \hbar\ \rightarrow$
q	$q\ \|\ \rightarrow$	$q\ \bullet\ \rightarrow$	$p\ \bullet\ \leftarrow$
r		$r\ 1\ \rightarrow$	$s\ \hbar\ \leftarrow$
(s)			

Partiendo de la entrada $||$, esta máquina de Turing hará:

$$\begin{aligned}
(p, \underline{|}\,|) &\vdash (q, \bullet\,\underline{|}) \vdash (q, \bullet\,|\,\underline{\hbar}) \\
&\vdash (p, \bullet\,\underline{|}\,\bullet) \\
&\vdash (q, \bullet\bullet\,\underline{\bullet}) \vdash (q, \bullet\bullet\bullet\,\underline{\hbar}) \\
&\vdash (p, \bullet\bullet\bullet\,\underline{\bullet}\bullet) \vdash (p, \bullet\bullet\,\underline{\bullet}\bullet\bullet) \vdash (p, \bullet\,\underline{\bullet}\bullet\bullet\bullet) \vdash (p, \underline{\bullet}\bullet\bullet\bullet\bullet) \\
&\vdash (p, \underline{\hbar}\bullet\bullet\bullet\bullet) \\
&\vdash (r, \underline{\bullet}\bullet\bullet\bullet) \vdash (r, |\,\underline{\bullet}\bullet\bullet) \vdash (r, ||\,\underline{\bullet}\bullet) \vdash (r, |||\,\underline{\bullet}) \vdash (r, ||||\,\underline{\hbar}) \\
&\vdash (s, ||||\,\underline{|})
\end{aligned}$$

momento en que quedará parada. Como s es un estado final, habrá *aceptado* la cadena $||$, y habrá dejado escrito en la cinta $||||$

La máquina se dirá determinista si, como en este caso, $|f(q,e)| \leq 1,\ \forall (q,e)$, es decir, dada una configuración, es posible a lo sumo una configuración siguiente.

Dada una información de entrada $x \in \Sigma^*$, la operación de la máquina de Turing puede producir:

1. que, tras una sucesión de pasos, la máquina se pare.

   a) si el estado en el que se encuentra entonces es final, habrá *aceptado* la cadena

   b) si el estado no es final, no habrá aceptado la cadena

   En la cinta habrá de todas formas una determinada información, cadena de $\gamma \in \Gamma^*$. Por lo tanto, ha calculado a partir de x un determinado valor $\gamma = \phi(x)$

2. que la máquina no consiga pararse. En este caso, desde luego no habrá aceptado la cadena de entrada, ni habrá calculado ningún valor

Concretamente, $MT1$ acepta todas las cadenas de $\{|\}^*$, y calcula la función $\phi : \{|\}^* \rightarrow \{|\}^*$ dada por $\phi(|^n) = |^{2n}$ para $n \geq 0$.

En general, una máquina de Turing:

- se para ante algunas cadenas, determinando un lenguaje

$$L_p(MT)\{x \in \Sigma^* / (q_0, x) \vdash^* (q, \gamma) \dashv \text{(parada)}\}$$

- define una función con llegada en Γ^* y origen en las cadenas del lenguaje anterior, es decir, una función

$$\phi : L_p \to \Gamma^*$$

, o sea, una función *parcial* $\phi : \Sigma^* \to \Gamma^*$

- Reconoce un lenguaje

$$L_r = \{x \in \Sigma^* / (q_0, x) \vdash^* (q, \gamma) \dashv \wedge\ q \in F\}$$

, sublenguaje del anterior.

Dicho de otro modo: dada una cadena $x \in \Sigma^*$, la máquina de Turing hace una y sólo una de estas tres cosas:

- se para ante x en un estado final: acepta la cadena y devuelve el valor $\phi(x) \in \Gamma^*$
- se para ante x en un estado no final: no acepta la cadena pero devuelve el valor $\phi(x) \in \Gamma^*$
- no se para ante x: ni acepta la cadena ni calcula un valor para $\phi(x)$

Puede ocurrir que L_r sea vacío o no, incluso que L_p sea vacío. En cualquier caso $L_r \subseteq L_p$

2.1.2. Capacidades de las máquinas de Turing

Las máquinas de Turing pueden usarse como se ha visto para reconocer lenguajes o para calcular funciones. Pero también para *generar* lenguajes. Por ejemplo

$\Gamma = \{0, 1, \hbar\}$ $\quad \Sigma = \{0, 1\}$
$q_0 \qquad\quad F = \emptyset$

f :

$MT2$	0	1	$\hbar$
q_0			$q_1\ 0\ \rightarrow$
q_1			$q_2\ \hbar\ \leftarrow$
q_2	$q_3\ 1\ \rightarrow$	$q_4\ 0\ \leftarrow$	
q_3	$q_3\ 0\ \rightarrow$	$q_3\ 1\ \rightarrow$	$q_2\ \hbar\ \leftarrow$
q_4	$q_3\ 1\ \rightarrow$	$q_4\ 0\ \leftarrow$	$q_3\ 1\ \rightarrow$

partiendo de la configuración $(q_0, \underline{\hbar})$ consigue, cada vez que pasa por el estado q_2, una cadena del lenguaje $0 \mid 1(0|1)^*$ (además en el orden que correspondería a la codificación de los enteros positivos expresados en base binaria 0, 1, 10, 11, 100 ...)

Sus capacidades son bastante amplias: pueden comparar un símbolo de entrada con uno concreto, y en función de ello pasar a un estado u otro, incluso comparar una cadena de una determinada longitud con otra y actuar según el resultado, y en particular buscar determinada cadena en la de de entrada. También pueden utilizar la cinta en forma de pila o de cualquier otra manera, bien para almacenar transitoriamente datos o para colocar resultados. Pueden recorrer la cadena de entrada a su antojo, copiarla más adelante, después de alguna marca de separación, etc. La imitación del comportamiento de un autómata finito es bien simple (usando la cinta sólo para la entrada, y con movimientos siempre a la derecha), o de un autómata con pila, poniendo una marca a la izquierda de la cadena de entrada, y usando toda la capacidad infinita de la izquierda para poner la pila. Pero tampoco tienen problema en reconocer, por ejemplo, si la cadena de entrada es de la forma $a^n b^n c^n$ [1].

Tienen memoria suficiente para "contar" o incluso para trabajar aritméticamente con cadenas que representen números.

[1] bastaría con empezar por el primer símbolo; si es una *a*, sustituirla por *A* y pasar hacia la derecha por todas las aes hasta encontrar una *b*; entonces se reescribiría esta *b* como *B*, y se pasaría hacia la derecha por más posibles bes hasta encontrar una *c*, reescribirla como *C*, y volver hacia la izquierda hasta encontrar de nuevo una *A*. Entonces reescribe esta *A* y se mueve a la derecha para repetir el proceso. Una vez que consiga pasar a mayúsculas todas las aes, bes y ces, aún debe ir a la derecha para comprobar que no hay nada detrás, o sea, hay un blanco, y entonces parar en un estado final. Si en cualquier momento de este proceso no encuentra lo que espera, se para, y entonces lo hará en un estado que no es de aceptación

En resumen, que las máquinas de Turing son en realidad "ordenadores" con un programa fijo (su función de transición) y con capacidad de almacenamiento infinita (la cinta). Lo demás es una cuestión de codificación. Todo lo que pueda hacerse con un ordenador y un determinado programa puede hacerse con una máquina de Turing. Como esto resulta incómodo y pesado, se eleva el nivel del lenguaje, de forma que se obtienen entornos más cómodos, como el dado por Pascal o C, por ejemplo. La ejecución de un programa Pascal que suponga capacidad de almacenamiento infinita, no es más que el funcionamiento de una máquina de Turing. Cada punto de ejecución del programa Pascal es un estado. Se transita de un estado a otro en función de los símbolos de entrada y posiblemente de informaciones intermedias que se hayan calculado y apuntado en algún lugar (de la cinta).

2.1.3. Ampliaciones y restricciones

En algunos casos conviene trabajar con máquinas de Turing restringidas. Por ejemplo, m.T. con cinta "semi-infinita", es decir, infinita sólo al lado derecho. Aunque puede parecer que estas máquinas tienen menos capacidad, no es cierto, por la misma razón que el conjunto de los enteros es numerable. Si se puede hacer algo con una m.T. con cinta doblemente infinita, se podrá hacer lo mismo con otra que tenga solamente una cinta semi-infinita, sin más que reservar las posiciones pares de ésta para la parte derecha de la cinta doblemente infinita y las impares para la parte izquierda. Habrá que modificar los avances y retrocesos (que serán de dos casillas cuando antes lo eran de una, y cambiarán de sentido según estemos en casilla par o impar), y algún detalle más.

Pueden verse muchas otras restricciones en los textos.

En cuanto a ampliaciones notables: tener 2 cintas no amplía las capacidades de una m.T. , aunque facilita el trabajo (ni en general tener n cintas).

Tampoco considerar M.T. no deterministas mejora la capacidad (vuelve a ocurrir como en los autómatas finitos).

2.1.4. Lenguajes recursivamente numerables y recursivos

Definición 2.1.4.1 *Un lenguaje L es* recursivo *si existe una máquina de Turing M tal que* $L_p = \Sigma^*$ *y* $L_r(M) = L$

Debe entenderse por lo tanto que la máquina de Turing que debe existir **debe pararse siempre**, ante cualquier cadena de entrada, sea o no de L; para las de L lo hará en un estado final y para las que no sean de L en un estado no final. Dicho de otro modo: un lenguaje es recursivo si se puede escribir un programa Pascal de la forma

```
program ReconoceL (input, output);
var x : string:
Begin
    readln (x);
    sus calculos, para saber si x esta o no en L
    if (le ha salido que x esta en L) then
        writeln ('¡Si!')
    else
        writeln ('No')
End.
```

o reformulado: existe un función Pascal de la forma `function fL (x:string):boolean` tal que $fL(x) = TRUE \Leftrightarrow x \in L$ (o, si apetece más, una función en C equivalente).

No es fácil imaginar ahora un lenguaje que no sea recursivo. Pero los hay (y muchos). El problema está en encontrar un **algoritmo** que lo caracterice.

Volviendo a la capacidad de las m.T. de generar lenguajes: supongamos que tenemos dos cintas; una de entrada y trabajo y otra exclusivamente de salida. El ejemplo $MT2$, podría modificarse un

poco, de forma que, cada vez que la máquina se encuentre en el estado q_2, antes de proseguir su trabajo, copiase en la cinta de salida la cadena que ha construido seguida de una marca, pongamos #. De esta manera se iría obteniendo la salida 0#1#10#11#100#101#... Por supuesto, la máquina no se pararía nunca, pero, cualquier cadena de $0 \mid (0|1)^*$ aparecería en la salida antes o después. A este comportamiento le llamaremos *generar* un lenguaje.

Definición 2.1.4.2 *Un lenguaje L es* recursivamente numerable *si existe una máquina de Turing capaz de generarlo.*

Dicho de otra manera, un lenguaje es recursivamente numerable si existe un programa en Pascal con el aspecto

```
program generaL (output);
var x : string ;
   ...
begin
     algunas operaciones (para conseguir la primera x)
     while (1=1) do
       begin
         writeln (x)
         algunas operaciones (para conseguir la siguiente x)
       end
end.
```

tal que, sucesivamente, va mostrando todas cadenas de L y sólo éstas. El programa no se para nunca, salvo que no sea capaz de conseguir una "siguiente" porque no la hay. Pero, si el lenguaje es infinito, no se puede generar entero en un tiempo finito.

Cada cadena de L debe aparecer en algún momento, pero no se exige que aparezca sólo una vez. Por ejemplo, si L es finito, el programa puede pararse cuando las haya escrito todas, o repetirlas todas o alguna indefinidamente, pero eso no importa.

Una formulación equivalente es la de poder disponer de un procedimiento `procedure siguienteEnL (var x : string)` capaz de, dada una cadena de L, devolver otra, también de L, de forma que la sucesión de llamadas al procedimiento las recorra todas a lo largo del tiempo.

Es evidente que,

Proposición 2.1.4.3 *Si un lenguaje es recursivo, entonces es recursivamente numerable.*

porque basta escribir el siguiente "programa" Pascal, suponiendo un alfabeto $A = \{a, b, c, \ldots\}$

```
program GeneraRec (output);
var x: string;
Begin
   x := '';
   while (1=1) do
     begin
         if fL(x) then writeln (x);
         x := sgte (x)
     end
End.
```

Aquí, el procedimiento `sgte` tiene la forma `procedure sgte (var x:string)` y modifica su parámetro para conseguir la siguiente cadena de la dada, en el orden natural. Por ejemplo, suponiendo un alfabeto $A = \{a, b, c, \ldots\}$, el orden es $\lambda, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa \ldots$, lo que no es mucho más difícil de lo que hacía $MT2$.

Dado un lenguaje recursivamente numerable, no puede garantizarse que sea recursivo, pero sí que existe un "casi algoritmo"[2] que "casi" lo reconozca:

[2] permítase la licencia

```
program CasiReconoce (input, output);
var x , y : string;
   ...
Begin
    readln (x);
    y := primera de L ;
    while (y <> x) do
      begin
          writeln ('Aun no lo se');
          siguienteEnL (y)
      end
    writeln ('¡Si!')
End.
```

El problema es que este "casi algoritmo" nunca es capaz de decir 'No'. Es posible, sin embargo, que el procedimiento de generación sea algo mejor, y sepa, por ejemplo, que sólo es capaz de generar cadenas de longitud par. Entonces, una pequeña modificación de `CasiReconoce` podría responder 'No' si *x* tiene longitud impar, pero eso no arreglaría mucho las cosas. Para obtener un verdadero algoritmo, deberíamos emplear otra técnica.

Resulta que

Proposición 2.1.4.4 *Si un lenguaje L es recursivamente numerable, existe una máquina de Turing capaz de responder afirmativamente a la pregunta de si una cadena está en L (y quizá negativamente en algunos casos en los que la cadena no lo esté).*

y viceversa:

Proposición 2.1.4.5 *Si una máquina de Turing es capaz de responder afirmativamente a la pregunta de si una cadena está en L (y quizá negativamente en algunos casos en los que la cadena no lo esté), entonces L es recursivamente numerable.*

Lo explica el siguiente algoritmo generador:

```
program GeneraUnRN (output);
procedure siguientePar (var i, j: integer);
  (* genera sucesivamente (1,1),(1,2),(2,1),(1,3),(2,2),(3,1),(1,4)...*)
  var s: integer;
  begin
    s:= i+j;
    if j=1 then
       begin i := 1; j := s end
    else
       begin i := i+1; j := j-1 end
  end;
var
Begin
  i:= 1; j:= 1;
  x := '';
  repeat
     for k := 1 to i-1 do sgte(x);
     haz j pasos de la M.T. (algoritmo "casi-reconocedor") de L ;
     if (ha dicho 'Si') then writeln (x);
     siguientePar (i,j)
  until (1=0)
End.
```

Hay que hacerlo así, porque es posible que con una cadena que no esté en el lenguaje la máquina "casi-reconocedora" no se pare, y no lleguemos ni siquiera a considerar la siguiente.

2.1.5. Propiedades de los lenguajes recursivos y recursivamente numerables

Proposición 2.1.5.1 *Si un lenguaje es recursivo, su complementario también*

Basta tomar el algoritmo de reconocimiento `ReconoceL` e intercambiar la salida ('Si' por 'No' y 'No por 'Si').

Proposición 2.1.5.2 *La unión de dos lenguajes recursivos es recursiva*

Una función de reconocimiento para $L_1 \cup L_2$, teniendo las funciones de reconocimiento respectivas $fL1$ y $fL2$ sería

```
function fL1UL2 (x: string):boolean;
begin
   readln (x);
   if fL1(x) then fL1UL2 := TRUE
   else if fL2(x) then fL1UL2 := TRUE
   else fL1UL2 := FALSE
end;
```

Proposición 2.1.5.3 *La intersección de dos lenguajes recursivos es recursiva*

Se prueba sencillamente usando la ley de De Morgan

Proposición 2.1.5.4 *La unión de dos lenguajes recursivamente numerables es recursivamente numerable*

Se puede construir un "algoritmo" "casi reconocedor" de la unión de dos recursivamente numerables L_1 y L_2 partiendo de los de éstos, a base de ejecutar *un paso* de cada uno, alternativamente. Si uno de los dos dice en algún momento 'Si', parar y devolver 'Si'. Con ello se garantiza la respuesta afirmativa exactamente en los elementos de la unión (que es lo requerido para demostrar que es recursivamente numerable).

Proposición 2.1.5.5 *Si un lenguaje es recursivamente numerable y su complementario también, entonces ambos son recursivos*

De forma parecida al caso anterior, se ejecuta un paso de la máquina "casi reconocedora" de L y un paso de la "casi reconocedora" de $\overline{L}$. Una de las dos tendrá que decir 'Si' en algún momento. Si ha sido la de L, se para dando 'Si'; si ha sido la de $\overline{L}$, sa para devolviendo 'No'. Por lo tanto, se para siempre, dando la respuesta de si $x \in L$ o no, así que ya se tiene un algoritmo reconocedor que prueba que L es recursivo (y por lo tanto, también $\overline{L}$)

Proposición 2.1.5.6 *Dado un lenguaje L, una de estas cuatro cosas es cierta (y sólo una)*

1. *L y $\overline{L}$ son recursivos*
2. *ni L ni $\overline{L}$ son recursivamente numerables*
3. *L es recursivamente numerable y no recursivo y $\overline{L}$ no es recursivamente numerable*
4. *$\overline{L}$ es recursivamente numerable y no recursivo y L no es recursivamente numerable*

Es consecuencia de las proposiciones anteriores.