13.3. MT para reconocer lenguajes

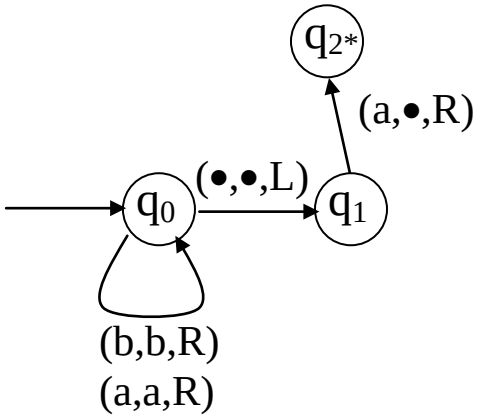
Gramática equivalente a una MT

Sea $M=(\Gamma,\Sigma,\bullet,Q,q_0,f,F)$ una Máquina de Turing. L(M) es el lenguaje aceptado por la máquina M.

A partir de M se puede crear una gramática generadora del tipo 0 de la jerarquía de Chomsky que genera el lenguaje L(M).

Ejemplo:

$M= (\{a,b,\bullet\},\{a,b\},\bullet,\{q_0,q_1,q_2\},q_0,f,\{q_2\})$
$f(q_0,b)=(q_0,b,R)$ $f(q_1,a)=(q_2,\bullet,R)$
$f(q_0,a)=(q_0,a,R)$
$f(q_0,\bullet)=(q_1,\bullet,L)$



Acepta el lenguaje descrito por (a+b)*a

Generamos la gramática:
$\Sigma_T=\Sigma=\{a,b\}$ $\Sigma_N=\{A,B,S,\bullet,[,],X,q_0,q_1,q_2\}$ con las siguientes reglas:

- Para generar x[q_0x] con x$\in${a,b}*:
  S::=X]
  X::=aAX | bBX | [q_0
  Aa::=aA Ba ::=aB Ab::=bA Bb::=bB
  A[q_0::=[q_0a B[q_0::=[q_0B

- Para simular el procesamiento de la máquina M:
- para cada transición f(p,b)=(q,c,R): pb::=cq
- para cada transición f(p,b)=(q,c,L): xpb::= qxc (para todo x$\in\Gamma$)

f(q_0,b)=(q_0,b,R) -> q_0b::= bq_0
f(q_0,a)=(q_0,a,R) -> q_0a::= aq_0
f(q_0,•)=(q_1,•,L) -> •q_0•::= q_1•• aq_0•::= q_1a• bq_0•::= q_1b•
f(q_1,a)=(q_2,•,R) -> q_1a::= •q_2

- [p::=[•p para todo p$\in$Q y p$\neq$ q_f
  -> [q_0::=[• q_0 [q_1::=[• q_1
- p]::= p•] para todo p$\in$Q y p$\neq$ q_f
  -> q_0]::=q_0•] q_1]::=q_1•]

- Para eliminar [xq_ky] donde q_k es el estado final:
- q_ky::= q_k (para todo y$\in\Gamma$)
  -> q_2a::= q_2 q_2b::= q_2 q_2•::= q_2
- xq_k::= q_k (para todo x$\in\Gamma$)
  -> aq_2::= q_2 bq_2::= q_2 •q_2::= q_2
- [q_k]::=λ
  -> [q_2]::=λ

$\Rightarrow$G es del tipo 0 y se puede demostrar que L(G)=L(M).

MT equivalente a una gramática de tipo 0

Sea $G=(\Sigma_T, \Sigma_N, S, P)$ una gramática generadora de tipo 0 y L(G) el lenguaje generado por G.

Es posible construir una máquina de Turing $M=(\Gamma,\Sigma,\bullet,Q,q_0,f,F)$ que acepta L(G).

Idea:

- La máquina se guarda la palabra de entrada en la cinta.
- En otro lugar de la cinta, la máquina simula la aplicación de las producciones de la gramática a la cadena inicial "S".
- Si hay varias posibilidades, la máquina prueba "en paralelo" todas ellas.
- En cada momento comprueba si una aplicación de una producción ha dado lugar a una cadena igual que la cadena de entrada. Si eso es así, termina en un estado final.

Teorema:

El conjunto de lenguajes que puedes ser aceptado por Máquinas de Turing y el conjunto de lenguajes que pueden ser generados por gramáticas generadoras es igual.
Este conjunto se denomina *Lenguajes recursivamente enumberables*.

(Capítulo 14.)

Máquinas de Turing y Computación

Tesis de Church/Turing

Las máquinas de Turing nos dan un modelo abstracto para computadores digitales.

Tesis de Church/Turing

"Cualquier computación que puede ser realizada por medios mecánicos puede ser realizada también con una máquina de Turing."

No es una afirmación que pueda ser probada, sino más bien una *definición* o *explicación* del concepto de *algoritmo*

Parece ser una definición adecuada:

- Otros formalismos propuestos de dicho concepto (funciones recursivas, sistemas de Post, sistemas de reescritura, etc.) no son más expresivos que las máquinas de Turing
- Nadie ha sugerido ningún problema que pueda ser resuelto por lo que intuitivamente se considera un algoritmo, y que no pueda ser implementado en una máquina de Turing

Una función f:E$\rightarrow$S se llama Turing *computable* (o simplemente *computable*), si existe una máquina de Turing M que computa f.

Máquina de Turing Universal

"Una máquina de Turing solo computa una función determinada, mientras los computadores digitales pueden realizar diferentes funciones (son programables)."

→ Se puede crear la máquina de Turing Universal (MUT):

Una MUT es una MT que dado como entrada la descripción de una máquina de Turing M y una cadena de símbolos w, puede simular la computación de M sobre w. Es decir, una MUT es una máquina generalizada y programable.

¿Cómo se puede codificar una MT?:

Consideramos las máquinas de Turing con las siguientes restricciones:

- alfabeto binario {0,1} más el blanco
- un único estado final q_2 (para el que f no esta definida)
- el estado inicial es siempre q_1

Se puede codificar cada transición:

- Ejemplo:
  $f(q_1,0)=(q_2,1,R)$:

  1 • 0 • 1 1 • 1 • 1

  q_1 0 q_2 b R (R->1, L->0)

Codificamos la máquina M como la concatenación de todas las cadenas correspondientes a las transiciones.

Funciones computables y no computables

Enumerabilidad de conjuntos

Definición:
Un conjunto (infinito) C es *enumerable*, sí existe una función biyectiva y total, $f:C \rightarrow N$ tal que f y f^{-1} son funciones computables.
(Si existe una correspondencia una-a-una de los elementos de C a los números naturales.)

Una *enumeración* de un conjunto C es una función sobrejectiva y total $g:N \rightarrow C$. Con ella se puede enumerar todos los elementos de C de forma $\{c_1, c_2, \ldots.\}$.

Ejemplos: (buscar la función f)
1.El conjunto de números enteros es enumerable.

0, –1, 1, –2, 2, –3, 3, –4, 4 ...
1 2 3 4 5 6 7 8 9 ...

2.El conjunto $N \times N$ es numerable.

(1,1), (1,2), (2,2), (2,1), (1,3), (2,3), (3,3), (3,2), (3,1) ...
1 2 3 4 5 6 7 8 9 ...

4.Los números reales entre 0 y 1 no son enumerables.

Teorema:
El conjunto de máquinas de Turing es enumerable.

Demostración:
Idea:

- Crear un algoritmo que genera todas las “codificaciones válidas” de máquinas de Turing
- Este algoritmo implementa la función de enumeración de la máquinas de Turing.

Funciones computables

Sin pérdida de generalidad consideramos sólo funciones con dominio y codominio N $(f:N \rightarrow N)$

De la tesis de Church se deriva que una función es computable si existe una máquina de Turing que la realiza.

Una función no computable

Sea T el conjunto de todas las funciones (Turing-)computables con dominio y codominio N.
Sea $\{f_1, f_2, f_3, ...\}$ una enumeración de T.

Teorema:

Sea $u(n) = \begin{cases} 1, & \text{si } f_n(n) \text{ no está definido} \\ f_n(n) + 1, & \text{en caso contrario} \end{cases}$.

La función u no es computable.

Demostración:

- Suponiendo que u es computable.
- Entonces debe existe una máquina de Turing que computa u.
- Es decir, en la enumeración de T hay un índice k con f_k=u.
- ¿Que valor tendrá u(k)?

$\rightarrow$ suponiendo que u es computable: f_k=u y por tanto :

$$u(k) = f_k(k)$$

$\rightarrow$ pero por la definición de u:

$$u(k) = \begin{cases} 1, & \text{si } f_k(k) \text{ no es definido} \\ f_k(k) + 1, & \text{en caso contrario} \end{cases}$$

$\rightarrow$ Se llega a una contradicción, y, por tanto u no puede estar en la enumeración de T: $f_1, f_2, f_3, ...$ Es decir, u no puede ser una función computable (no existe una máquina de Turing para computar u).

Problemas de decisión

Definición:
Un *problema de decisión* consiste en determinar si una determinada afirmación respecto a los elementos de un conjunto es cierta o falsa

Ejemplo:
"Dado un número n. ¿Es n un número primo?"

Definición:
Sea C un conjunto y P un problema de decisión sobre los elementos de C.
Se llama *función de decisión del problema* a la función $f:C\rightarrow\{0,1\}$, donde:

$$f(w)=\begin{cases}1, \text{si w tiene la propiedad R}\\ 0, \text{en caso contrario}\end{cases}$$

para todo $w\in C$.

Definición:
Un problema se llama *decidible* si su función de decisión es computable (existe una máquina de Turing que computa f). Un problema es *indecidible* si no es decidible.

Algunos problemas indecidibles:

- Reconocimiento de palabras por máquinas de Turing: Dada una palabra w y una MT ¿MT acepta w?
- Dada una MT y una entrada w ¿Para o no para MT con w?
- Dada una gramática independiente del contexto G, ¿Es G ambigua?
- Dado un programa en C++: ¿El programa ejecutará un bucle infinito o no?
- Dado dos programas P1 y P2 en Java: ¿ P1 y P2 calculan lo mismo?
- …