Febrero 2018

viernes, 6 de enero de 2023 20:29

EJERCICIO 1 (1,5 puntos)

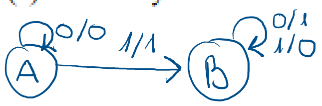
Dado un número natural *A* expresado en notación binaria con *n* bits, el cambio de signo de *A* (es decir, -*A*) se calcula por medio de la operación "Complemento a 2". Una forma sencilla de calcular el complemento a dos de un número binario es comenzar por la derecha (el dígito menos significativo), copiando el número original (de derecha a izquierda) hasta encontrar el primer 1, después de haber copiado el 1, se niegan (complementan) los dígitos restantes (es decir, copia un 0 si aparece un 1, o un 1 si aparece un 0). Por ejemplo, el complemento a dos de «0011 11010» es «1100 00110».

(a) Desarrolle la operación "Complemento a 2" por medio de un Autómata de Mealy.

(b) Desarrolle la operación "Complemento a 2" por medio de un Autómata de Moore.

(c) Enuncie y demuestre el Lema de Bombeo para Autómatas Finitos.

a)

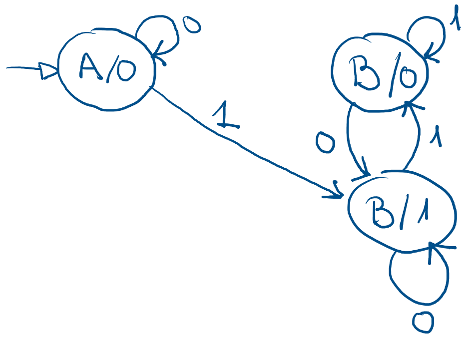


← $\boxed{\text{Autómata de Mealy}}$

Estado A: Copia el número original (de derecha a izquierda) hasta encontrar el primer 1. Después de haberlo copiado transiciona al estado B.

Estado B: Niega los dígitos restantes [(0→1), (1→0)]

b)



← $\boxed{\text{Autómata de Moore}}$

Estado A/0: Copia el nº original de derecha a izquierda. Cuando lee el primer 1 lo copia y pasa al estado B/1

Estado B/1: Invierte el 0 y escribe un 1.

Estado B/0: Invierte el 1 y escribe un 0.

EJERCICIO 2 (1,5 puntos)

Considere la siguiente gramática libre de contexto, expresada en Forma Normal de Chomsky.

S → B L
S → **id**
B → **lbracket**
L → S Q
Q → C L
Q → **rbracket**
C → **comma**

Verifique que la cadena "**[[id , id] , id , [id]]**" pertenece al lenguaje definido por la gramática por medio del algoritmo de Cocke-Younger-Kasami.

[	[	id	,	id	]	,	id	,	[	id	]	]
B	—	—	—	—	—	—	—	—	—	—	—	S*
	B	—	—	—	S	—	—	—	—	—	—	L
		S	—	—	L	—	—	—	—	—	—	—
			C	—	Q	—	—	—	—	—	—	—
				S	L	—	—	—	—	—	—	—
					Q	—	—	—	—	—	—	—
						C	—	—	—	—	—	Q
							S	—	—	—	—	L
								C	—	—	—	Q
									B	—	S	L
										S	L	—
											Q	—
												Q

Como se puede generar la cadena a partir del símbolo inicial de la gramática, la cadena pertenece al lenguaje.

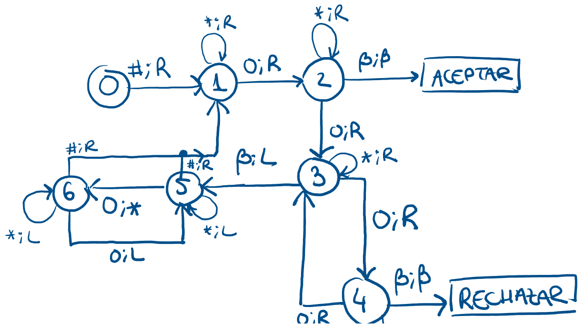
EJERCICIO 3 (2 puntos)

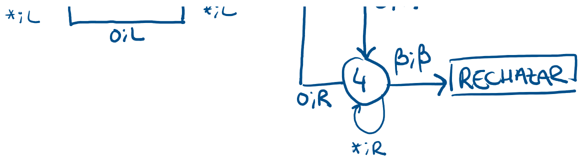
Desarrolle una Máquina de Turing que reconozca el lenguaje formado por cadenas de 0 cuya longitud sea potencia de dos:

$$L = \left\{ 0^{2^n};\ n \geq 0 \right\}$$

2
4
8
16
32
⋮

NOTA: Para comprobarlo hay que hacer varias pasadas a la cadena eliminando la mitad de 0´s en cada pasada hasta alcanzar una cadena de longitud 1. Si en alguna pasada el número de 0´s es impar hay que rechazar la cadena.





En primer lugar leemos el símbolo # y nos desplazamos a la derecha para comenzar a leer la cadena.

En el estado 1 recorremos la cadena hacia la derecha cuando leemos el primer 0 pasamos al siguiente estado.

En el estado 2 se ha leído 1 cero ($2^0 = 1$) por lo que si termina la cadena, se acepta.

Si se encuentra un 0 pasa al estado 3 que indica que el número de 0's es par. Si encuentra otro cero será impar y pasará al estado 4. Si termina con un número impar de ceros entonces se rechaza la cadena.

Si el número es par pasa al estado 5 donde se eliminan la mitad de los ceros (el $5 \to 6$ elimina y del $(6 \to 5)$ lo salta.

Si se encuentra el inicio de la cadena salta al estado 1 y se vuelve a realizar el procedimiento.

EJERCICIO 4 (1 punto)

Sea EQ_{TM} el lenguaje formado por las cadenas $<M_1,M_2>$ tales que M_1 y M_2 son codificaciones de Máquinas de Turing que reconocen el mismo lenguaje. Es decir, $L(M_1) = L(M_2)$.

Demuestre que el lenguaje EQ_{TM} es indecidible.

NOTA: Considere demostrado que los lenguajes A_{TM} (problema de la aceptación), $HALT_{TM}$ (problema de la parada) y E_{TM} (problema del lenguaje vacío) son indecidibles.

Podemos crear una máquina de Turing, M_1, que rechace todas las entradas. De esta forma aceptaría solamente el lenguaje vacío.

Suponemos EQ_{TM} decidible y, por lo tanto, existe una máquina R capaz de decidir el lenguaje.

Utilizando R y M_1 podemos construir S tal que:

$$S(<M>) = \begin{cases} \text{acepta} & \text{si } R(<M_1, M_2>) \text{ acepta} \\ \text{rechaza} & \text{si } R(<M_1, M_2>) \text{ rechaza} \end{cases}$$

La máquina S soluciona el problema E_{TM} y como utiliza la máquina R, ninguna de las dos máquinas puede ser construida.

Como no se puede construir una máquina que reconozca el lenguaje, el problema EQ_{TM} es indecidible.

EJERCICIO 5 (2 puntos)

Considere el modelo de computación de las funciones recursivas. Asuma que las siguientes funciones ya han demostrado ser recursivas primitivas: *Suma(x,y), Producto(x,y), Potencia(x,y), Decremento(x), RestaAcotada(x,y), Signo(x), SignoNegado(x), Min(x,y), Max(x,y), And(x,y), Or(x,y), Not(x), Igual(x,y), Mayor(x,y), Menor(x,y), MayorOIgual(x,y), MenorOIgual(x,y), If(x,y,z).*

Demuestre que la función *Raiz*(*x*,*n*), que calcula la raíz n-esima de un número entero, es una función primitiva recursiva.

$$Raiz(x,n) = \lfloor \sqrt[n]{x} \rfloor = y \mid y^n \leq x < (y+1)^n$$

	X	n	Y	
	0	3	0	
	1	3	0	
	2	3	0	
$3^1 \leftarrow$	3	3	1	+1
	4	3	1	
	5	3	1	
	6	3	1	
	7	3	1	
	8	3	1	
$3^2 \leftarrow$	9	3	2	+1
	10	3	2	

CASO BASE

$Raiz(0, n) = Z_1$

CASO GENERAL

$Raiz(S(x), n) = g(Raiz(x,n), x, n) =$

$= g(U_1^3, U_2^3, U_3^3)$

Podemos observar que el resultado sigue un patrón de comportamiento que podemos definir como:

$S(Raiz(x,n))$ si $Potencia(U_3^3, S(U_1^3)) \geq S(U_2^3)$

$$\mathrm{Raiz}(S(x),n) = \begin{cases} S(\mathrm{Raiz}(x,n)) & \text{si} \quad \mathrm{Potencia}(U_3^3,\ S(U_1^3)) \geq S(U_2^3) \\ \\ \mathrm{Raiz}(x,n) & \text{en otro caso} \end{cases}$$

Por lo tanto la función recursiva sería:

$$\mathrm{Raiz}(S(x),n) = R(Z_1, C(\mathrm{If}(C(C(\mathrm{MayorIgual}, C(\mathrm{Potencia}, U_3^3, C(S, U_1^3)), C(S, U_2^3)), \\ , C(S, U_1^3), \\ , U_1^3)))$$

EJERCICIO 6 (1 punto)

¿Qué es un verificador de un lenguaje? Demuestre que un lenguaje es NP si y solo si es verificable polinomialmente.

Un verificador de un lenguaje A es un algoritmo que permite conocer si una cadena w pertenece al lenguaje. Si su complejidad temporal depende polinómicamente de la longitud de la cadena, se dice que es un verificador en tiempo polinomial.

Si un lenguaje, A, es NP entonces existe una máquina de Turing no determinista que lo decide en un tiempo polinomial. Podemos construir un verificador en tiempo polinomial de la siguiente forma:

- Dado $\langle w, c \rangle$
  1- Ejecutar la máquina sobre w tratando cada símbolo de c como la elección de cada paso indeterminista.
  2- Si el camino seleccionado por c acepta w entonces acepta, rechaza en caso contrario.

Si un lenguaje es verificable polinomialmente entonces existe un verificador V en tiempo polinomial. Podemos construir una máquina de Turing indeterminista existe un verificador V en tiempo polinómico.

Podemos construir una máquina de Turing indeterminista de la siguiente forma:

1 - Generar de forma indeterminista una cadena c de tamaño n^a.

2 - Ejecutar V sobre $<w, c>$

3 - Si V acepta → acepta
si no → rechaza.