Septiembre 2018

jueves, 5 de enero de 2023 19:50

EJERCICIO 1 (1,5 puntos)

(a) ¿Qué es una Autómata de Pila?

(b) ¿Qué diferencia hay entre un Automata de Pila Determinista e Indeterminista?

(c) ¿Tienen la misma capacidad? Razone la respuesta.

a) Un autómata de pila es una sextupla $(\Sigma, \Gamma, Q, \Delta, q_0, F)$ donde:

- Σ: es el alfabeto de la entrada + el símbolo β.
- Γ: es el alfabeto de la pila + γ (pila vacía)
- Q: es el conjunto de estados
- Δ: es el conjunto de transiciones
- q_0: es el estado inicial
- F: es el conjunto de estados finales.

b) Los autómatas de pila deterministas tienen, en cada momento, una posible transición. En otro caso sería un autómata no determinista.

c) Los APD no tienen la misma capacidad que los APND ya que el APND puede estar computando varios caminos simultáneamente. Es imposible simular dicho estado simultáneo en un APD porque no podríamos tener una única pila para cada estado del APND.

EJERCICIO 2 (1,5 puntos)

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

S → **number**
S → **id**
S → L N
N → B R
L → **lparen**
R → **rparen**
B → S B
B → **number**
B → **id**
B → L N

Verifique que la cadena "(a (b (2)) (c))" pertenece al lenguaje definido por la gramática por medio del algoritmo de Cocke-Younger-Kasami

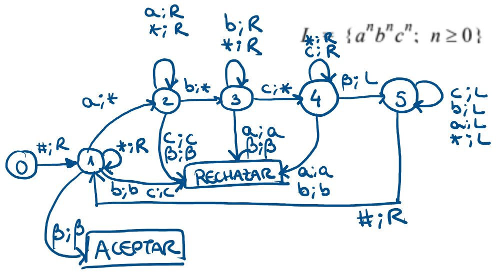
(	a	(	b	(	2	)	)	(	c	)	)
L	—	—	—	—	—	—	—	—	—	—	S*
	S,B	—	—	—	—	—	B	—	—	B	N
		L	—	—	—	—	S,B	—	—	B	N
			S,B	—	—	B	N	—	—	—	—
				L	—	S,B	N	—	—	—	—
					S,B	N	—	—	—	—	—
						R	—	—	—	—	—
							R	—	—	—	—
								L	—	—	N
									S,B	S,B	—
										N	—
										R	R

Como se puede obtener 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 siguiente lenguaje:

$L = \{a^n b^n c^n;\ n \geq 0\}$



En primer lugar lee el símbolo # para comenzar a leer la cadena.

En el estado 1 comprueba el primer símbolo. Si es β entonces $n=0$ y lo acepta

Si es una b o una c entonces lo rechaza

Si es una a lo marca y avanza hacia el siguiente. Si el siguiente es una c o un β entonces lo rechaza.

Si es una b entonces lo marca y avanza al siguiente caracter distinto de b y *.

Si es una a o un β lo rechaza.

Si es una c lo marca y avanza hasta el siguiente. Si no es un β, lo rechaza. Si es un β se mueve hacia la izq. hasta encontrar # y recorre de nuevo la cadena saltándose los *.

EJERCICIO 4 (1,5 puntos)

Sea E_{TM} el lenguaje formado por las cadenas <M> tales que M es la codificación de una máquina de Turing que no reconoce ninguna entrada, es decir, cuyo lenguaje es el lenguaje vacío. Demuestre que el lenguaje E_{TM} es indecidible.

NOTA: Considere demostrado que los lenguajes A_{TM} (problema de la aceptación), $HALT_{TM}$ (problema de la parada).

y recorre de nuevo la cadena saltándose los *.

EJERCICIO 4 (1,5 puntos)

Sea E_{TM} el lenguaje formado por las cadenas <M> tales que M es la codificación de una máquina de Turing que no reconoce ninguna entrada, es decir, cuyo lenguaje es el lenguaje vacío. Demuestre que el lenguaje E_{TM} es indecidible.

NOTA: Considere demostrado que los lenguajes A_{TM} (problema de la aceptación), $HALT_{TM}$ (problema de la parada).

Demostración por reducción

Dada una entrada w y la máquina M podemos construir otra máquina M_1 tal que

$$M_1(x) = \begin{cases} \text{rechaza} & \text{si } x \neq w \\ \text{ejecuta } M(w) \text{ y acepta si } M \text{ acepta} & \text{si } x = w \end{cases}$$

Suponemos E_{TM} decidible y, por lo tanto, existirá una máquina R que lo decida. A partir de M_1 y R podemos construir una máquina S tal que:

1º Construye M_1
2º Aplique $R<M_1, w>$
3º Acepta si R acepta
4º Rechaza si R rechaza.

S soluciona el problema de aceptación sobre máquina de Turing por lo que no puede existir y, por lo tanto, tampoco R. Como el lenguaje no puede ser reconocido no es decidible.

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 *Log2(x+1)*, que calcula el logaritmo en base 2 de un número entero, es una función primitiva recursiva.

NOTA: El logaritmo está definido para números mayores o iguales a 1. Al utilizar el argumento (x+1) el caso base de la recursión es x=0.

$$Log2(x+1) = y \mid 2^y \leq x+1 < 2^{y+1}$$

$x+1$	$\log 2(x+1)$
1	0
2	1 }+1
3	1
4	2 }+1
5	2
6	2
7	2
8	3 }+1
9	3
⋮	⋮

CASO BASE

$$\log 2(1) = Z_0$$

CASO GENERAL

$$\log 2(S(x+1)) = g(\log 2(x+1), x+1) = g(U_1^2, U_2^2)$$

Podemos observar que el resultado de la función sigue un patrón de comportamiento

$$\log 2(S(x+1)) = \begin{cases} S(\log 2(x+1)) & \text{si } 2^{S(y)} \geq S(x+1) \\ \log 2(x+1) & \text{en otro caso} \end{cases}$$

Para generar el número 2 tendríamos que hacer $S(S(Z_2))$
Por tanto la función recursiva quedaría como

$$\mathrm{Log2}(x+1) = R(Z_0, C(\mathrm{If}, C(\mathrm{MayorIgual}, C(\mathrm{Potencia}, C(S, C(S, Z_2)), C(S, U_1^2)), C(S, U_2^2)), C(C(S, U_1^2)), U_1^2))).$$

EJERCICIO 6 (1,5 puntos)

¿Qué es un problema NP-completo? Enuncie el Teorema de Cook y Levin y describa brevemente su demostración.

Un problema NP-completo es un problema que pertenece a la clase NP y que, si tuviera solución en tiempo polinomial, permitiría demostrar que todos los problemas NP tienen solución en tiempo polinomial.

TEOREMA – COOK Y LEVIN

Sea SAT el lenguaje formado por las fórmulas booleanas satisfactibles, es decir, que existan valores que hacen cierta a la fórmula.

El teorema de Cook-Levin establece que el lenguaje SAT pertenece a la clase P sí y solo sí $P = NP$. es decir, SAT es un problema NP-completo y, por tanto existe un verificador que verifica SAT en tiempo polinomial. (tabla de verdad).

El teorema plantea un tablero de configuración en el que si se alcanza un estado de aceptación significaría que la máquina acepta w.

La idea de Cook y Levin era convertir el tablero en un conjunto de fórmulas booleanas de orden $O(n^{2k})$, por lo tanto se demostraría que cualquier problema NP puede reducirse en tiempo polinomial a un problema de Satisfactibilidad (SAT) y, por lo tanto, SAT sería NP-completo.