Tema 1

Febrero 2016
EJERCICIO 1 (2 puntos)
¿Qué significa que un conjunto es numerable o contable? Indique dos ejemplos de conjuntos numerables y dos ejemplos de conjuntos no numerables.

Se dice que un conjunto A es numerable o contable si es posible construir una correspondencia biunívoca entre el conjunto A y el conjunto de los números naturales o un subconjunto de los números naturales.

Ejemplos numerables
- Números enteros $\mathbb{Z}$
- Parejas de números naturales ($\mathbb{N}$, $\mathbb{N}$)

Ejemplos no numerables:
- Número reales $\mathbb{R}$
- La cardinalidad de $\mathbb{R}$.

Febrero 2014
EJERCICIO 1
¿Qué es "computar"? ¿Qué es un "modelo de computación"? ¿Qué aspectos hay que considerar al definir un modelo de computación?

Se define "computar" como realizar un cálculo matemático por medio de un conjunto finito de operaciones elementales.

Un modelo de computación es un modelo abstracto que describe una forma de computar

Los aspectos a considerar al definir un modelo de computación son:
– ¿Cómo se representan las entradas?
– ¿Cómo se representan las salidas?
– ¿Cuáles son las operaciones elementales?
– ¿Cómo se combinan las operaciones para desarrollar el "programa"?

Febrero 2014
EJERCICIO 2
Considere los modelos de computación dedicados a desarrollar funciones sobre conjuntos finitos. ¿Qué es un programa lineal? ¿Qué elementos aparecen en su definición?

Un programa lineal es una secuencia de instrucciones del tipo
– Entrada: (s READ x)
– Salida: (s OUTPUT i)
– Operación: (s OP i … k)
Donde s indica el número de instrucción, x es una variable de entrada, OP es una operación elemental, i … k representan los resultados de las instrucciones i-ésima … k-ésima y se cumple que s > i … k.

Tema 3

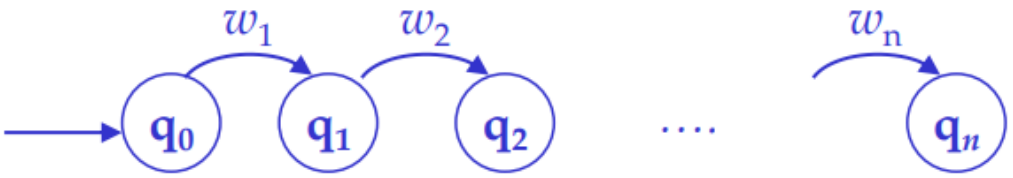
Septiembre 2020
EJERCICIO 1 (1.5 puntos)
Esta pregunta aparece en Septiembre 2019 EJERCICIO 1 (1 punto)
Esta pregunta aparece en Febrero 2017 EJERCICIO 1 (1 punto)
(a) Enuncie y demuestre el Lema de Bombeo para Autómatas Finitos.

Sea L un lenguaje regular sobre el alfabeto Σ reconocido por un DFSM con m estados. – Si w L y |w|≥m, entonces existen las cadenas r, s y t con |s|≥1 y |rs|≤m tales que w = rst y para todo n ≥ 0, rsn t también pertenece a L.
En término de funciones, la función reconocida por un DFSM de m estados f: Σ*Ψ* cumple que,

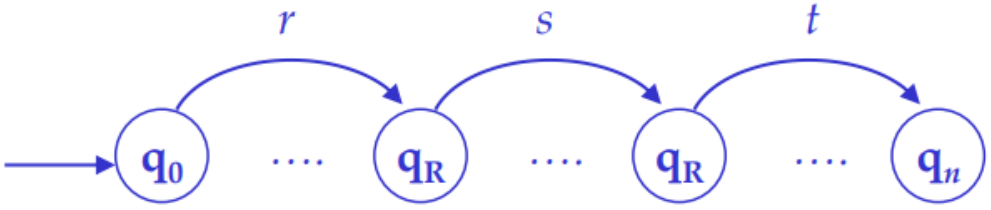
- $\forall w \in \Sigma^*, |w| \geq m$ y $f(w)=a$,
- $\exists\, r, s, t, b, c, d$ con $|s| \geq 1$, $|rs| \leq m$,
- tales que $w = rst$, $a=bcd$
- y $\forall n \geq 0, f(rs^n t) = bc^n d$

Identifica el tipo de funciones que se pueden definir por medio de autómatas finitos. Las funciones que no lo cumplan no pueden ser descritas por autómatas finitos.

Demostración:
Siguiendo el comportamiento del DFSM, cada símbolo de la cadena de entrada w genera una transición entre estados:



Teniendo en cuenta que existen m estados diferentes y que |w|≥m, entonces debe existir al menos un estado qR repetido en el conjunto de transiciones. Esto se conoce como "pigeonhole principle" (principio del palomar): Si en un palomar hay n huecos y n+1 palomas, en algún hueco hay más de una paloma.

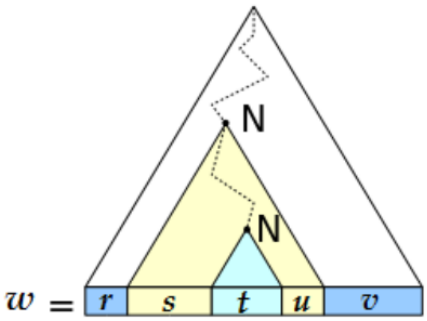


(b) Enuncie y demuestre el Lema de Bombeo para Autómatas de Pila.

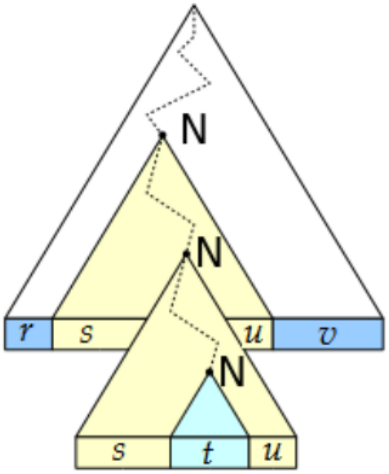
Si L es un lenguaje libre de contexto, existe un p1 tal que, toda cadena del lenguaje w de longitud mayor que p (|w|>p) puede escribirse como w = r s t u v, donde las subcadenas cumplen que:

- $| s\ t\ u | \leq p$
- $| s\ u | \geq 1$
- $r\ s^n\ t\ u^n\ v \in \mathrm{L}$

Demostración:
Puesto que la gramática libre de contexto tiene un número de símbolos no terminales finito, si una cadena w es lo suficientemente larga, entonces en su árbol de derivación debe repetirse algún símbolo no terminal N.



Esto permite "bombear" la derivación de N a N



Septiembre 2018 y Septiembre 2017
EJERCICIO 1 (1,5 puntos)
(a) ¿Qué es un Autómata de Pila?

Un autómata de pila (pushdown automaton – PDA) se define como una sextupla (Σ, Γ, Q, Δ, q_0, F) donde:
- Σ es el alfabeto de entrada, incluido el símbolo vacío β.
- Γ es el alfabeto de la pila, incluido el símbolo vacío γ.
- Q es el conjunto de estados del autómata.
- Δ es el conjunto de transiciones
- q0 es el estado inicial del autómata
- F es el conjunto de estados finales del autómata.

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

Un autómata de pila es determinista si en cada momento solo es posible realizar una transición. En caso contrario es un autómata no determinista.
– Formalmente, un autómata de pila es determinista si cumple las siguientes condiciones:
1. Para cada q Q, a Σ ∪ { } , x Γ, solo existe una o ninguna transición (q, a, x; p, z)
2. Para cada q Q, x Γ , si existe la transición (q, , x; p, z) entonces no existen transiciones (q, a, x; p, z).

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

Tendrían la misma capacidad si para todo APND existiera un APD equivalente. Sin embargo:
Los deterministas (APD) y los no deterministas (APND) no tienen la misma capacidad de cómputo.
– El APND puede estar computando diferentes caminos simultáneamente.
– En cada momento, la configuración del APND corresponde al estado de la unidad de control y al contenido de la pila.
– En este caso es imposible simular el "estado simultáneo" (formado por varios estados y varias pilas) por medio de un único estado y una única pila. El problema está en que no es posible simular varias pilas con una única pila.
– Por tanto, dado un APND es imposible obtener un APD equivalente.

Tema 4

Febrero 2014
EJERCICIO 6
¿Qué es una máquina universal de Turing?

Una Máquina de Turing está formada por una unidad de control, que puede describirse mediante un autómata finito, una cinta de lectura y escritura, que tiene un comienzo a la izquierda y se extiende indefinidamente hacia la derecha, y un cabezal, que indica la posición de la cinta sobre la que trabaja la máquina en cada paso.

Tema 6

Febrero 2014

EJERCICIO 8

En el modelo de computación de funciones recursivas, se considera computable toda función que se pueda construir a partir de ciertas funciones básicas y ciertos mecanismos de combinación. ¿Cuáles son esas funciones básicas? ¿Cuáles son los mecanismos de combinación?

Funciones básicas

- Función cero:

$$Z_k(n_1, n_2, \ldots, n_k) = 0$$

- Función sucesor:

$$S(n) = n+1$$

- Función proyección:

$$U_i^k(n_1, n_2, \ldots, n_k) = n_i$$

Composición de funciones:

Dadas f, una función de aridad k, y $g_1,\ldots,g_k$, funciones de aridad n, se define la **composición de f con $g_1,\ldots,g_k$** como la función

$$C(f, g_1,\ldots,g_k) = h(x_1,\ldots,x_n) = f(g_1(x_1,\ldots,x_n), \ldots, g_k(x_1,\ldots,x_n))$$

Tema 7

Septiembre 2020, Septiembre 2019 y Febrero 2019
EJERCICIO 6 (1 punto)
(a) ¿Qué es un lenguaje NP?

Un lenguaje verificable polinomialmente.

Aparece también en Febrero 2018 Ejercicio 6
(b) ¿Qué es un verificador de un lenguaje?

Un verificador de un lenguaje A es un algoritmo V tal que
A = { w | V acepta <w,c> para alguna cadena c }
es decir, w pertenece a A si existe una cadena c tal que el algoritmo V acepta <w,c>.

$$w \in A \iff \exists\, c \mid V(w,c) \text{ se acepta}$$

- Se dice que V es un verificador en tiempo polinomial si su complejidad temporal depende polinomialmente de la longitud de w.
- Se dice que un lenguaje A es verificable polinomialmente si existe un verificador en tiempo polinomial para ese lenguaje.

Aparece también en Febrero 2018 Ejercicio 6
(c) Demuestre que un lenguaje es NP si y solo si es verificable polinomialmente.

Demostración: A es NP => es verificable polinomialmente
Si A es verificable polinomialmente, entonces existe un verificador V en tiempo polinomial (n^k). Podemos construir una Máquina de Turing indeterminista en tiempo polinomial de la siguiente forma:
Dada w de longitud n:
1.- Generar de forma indeterminista una cadena c de longitud máxima n^k.
2.- Ejecutar V sobre <w,c>
3.- Si V acepta, aceptar. Si no, rechazar.

Demostración: A es NP <= es verificable polinomialmente
Si A es NP entonces existe una Máquina de Turing no determinista (MT) que decide en tiempo polinomial $O(n^k)$. Podemos construir un verificador V en tiempo polinomial $O(n^k)$ de la siguiente forma:
Dadas <w,c> :
1.- Ejecutar MT sobre w, tratando cada símbolo de c como la elección de cada paso indeterminista de MT.
2.- Si el camino seleccionado por c acepta w, aceptar. Si no, rechazar.

Febrero 2020, Septiembre 2018 y Septiembre 2017
EJERCICIO 6 (1.5 punto)
(a) ¿Qué es un problema NP-completo?

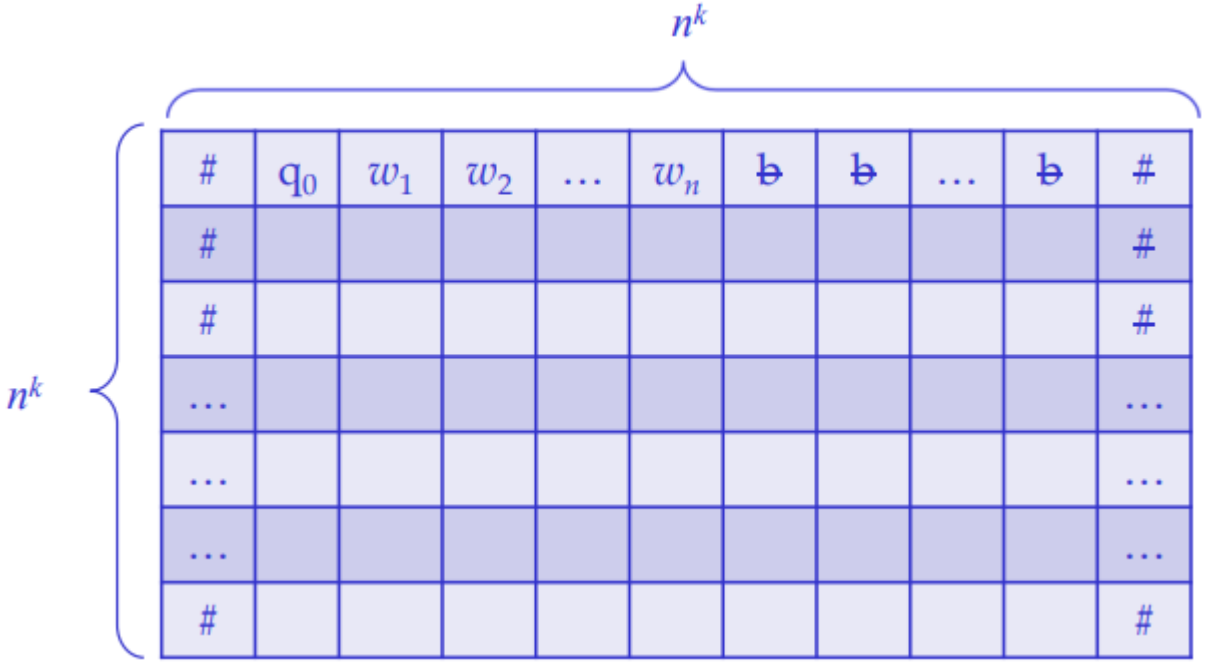
Existen una serie de problemas pertenecientes a NP que, si tuvieran solución en tiempo polinomial, permitirían demostrar que todos los problemas NP tienen solución en tiempo polinomial. Estos problemas se denominan NP-completos

(b) Enuncie el Teorema de Cook y Levin y describa brevemente su demostración.

Sea SAT el lenguaje formado por las fórmulas booleanas satisfacibles. El teorema de Cook-Levin establece que

$$SAT \in P \Leftrightarrow P = NP$$

Dado un lenguaje A NP, existe una Máquina de Turing no determinista, N, que resuelve el lenguaje en tiempo polinomial n^k. Es decir, para una entrada w de longitud n el número máximo de iteraciones de N es n^k. Vamos a definir un tablero basado en las configuraciones del proceso de ejecución de N:



Cada fila del tablero contiene una configuración de la máquina N. La primera fila contiene la configuración inicial y las siguientes filas corresponden a una transición de la máquina. Un tablero se acepta si contiene una configuración que alcance el estado de aceptación. Un tablero aceptado corresponde a una ejecución de la máquina N que conduce a un estado de aceptación.
Calcular si la máquina N acepta una entrada w equivale a calcular si existe un tablero aceptable.

Vamos a representar el problema de encontrar un tablero aceptable como un problema de satisfacción de una fórmula lógica Sea Q el conjunto de estados de N y Σ el alfabeto del lenguaje. El conjunto de símbolos que puede aparecer en cada celda del tablero es C = Q Σ {#,b}. El tamaño de C no depende de la longitud de la entrada (n). Se define la variable booleana xijc como una variable que indica que en la celda (i,j) se encuentra el símbolo c.
La fórmula lógica asociada al tablero completo está formada por cuatro partes

$$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{accept}.$$

La fórmula φcell corresponde a la exigencia de que para cada celda (i,j) una y solo una de las variables booleanas xijc puede ser cierta.

$$\phi_{cell} = \bigwedge_{i,j}((\bigvee_{c} x_{ijc}) \wedge (\bigwedge_{s \neq t}(\overline{x_{ijs}} \vee \overline{x_{ijt}})))$$

Teniendo en cuenta que el número de celdas es $(n^k)^2$, este término es de orden $O(n^{2k})$.
La fórmula φstart corresponde a la descripción de la primera fila:

$$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,b} \wedge \ldots \wedge x_{1,}$$

Este término es de orden $O(n^k)$.
La fórmula φmove corresponde a la descripción de los valores aceptables de trozos de 3x2 celdas. Esto permite describir las transiciones de la máquina de Turing. Por ejemplo, los siguientes grupos podrían ser aceptables:

(a)

a	q_1	b
q_2	a	c

(b)

a	q_1	b
a	a	q_2

(c)

a	a	q_1
a	a	b

(d)

#	b	a
#	b	a

(e)

a	b	a
a	b	q_2

(f)

b	b	b
c	b	b

Este término es de orden $O(n^{2k})$, aunque la constante pueda ser muy grande (como mucho c^6).
La fórmula φaccept indica que para que un tablero se acepte, al menos una celda debe corresponder al estado de aceptación:

$$\phi_{accept} = \bigvee_{i,j} x_{i,j,q_{accept}}$$

Este término es de orden $O(n^{2k})$
Por tanto, cualquier problema NP puede transformarse en buscar una solución al problema SAT sobre una fórmula lógica φ. Como la fórmula lógica es de orden $O(n^{2k})$, esto quiere decir que cualquier problema NP puede reducirse en tiempo polinomial a un problema SAT. Por tanto, SAT es NP-completo.

Tema 8 NO ENTRA

Septiembre 2020
EJERCICIO 7 (1 punto)
(a) Enuncie el Teorema de Savitch.

Toda máquina de Turing indeterminista con una complejidad espacial de orden f(n) ≥ n puede convertirse en una máquina de Turing determinista con una complejidad espacial de orden f^2 (n).

En Febrero 2018 Ejercicio 7, además se pide:
Describa brevemente su demostración

No es posible demostrar el teorema considerando una máquina de Turing determinista que explore de forma consecutiva todos los caminos de la máquina de Turing indeterminista. La razón es que si el espacio utilizado es de orden O(f(n)), el tiempo de ejecución puede llegar a ser de orden $O(2^{f(n)})$ y serían necesarias $O(2^{f(n)})$ celdas para almacenar cada posible ramificación.

La demostración se basa en lo que se conoce como yieldability problem o problema de la capacidad de rendimiento, que consiste en estudiar si desde una configuración c_1 se puede alcanzar otra configuración c_2 en t pasos. CANYIELD(c_1, c_2 , t)
1. Si t=1, verificar si c_1 = c_2 o si se puede alcanzar c_2 desde c_1 con una única transición de la máquina de Turing indeterminista.
2. Si t>1, para cada configuración cm de N que ocupe un espacio f(n)
    1. Ejecutar CANYIELD(c_1 , cm, t/2)
    2. Ejecutar CANYIELD(cm, c_2 , t/2)
    3. Si las dos ejecuciones se aceptan, aceptar
3. Si el bucle anterior no llega a aceptar, rechazar

El algoritmo CANYIELD() es muy ineficiente en cuanto a tiempo. La búsqueda exhaustiva de c_m es de orden exponencial $O(2^{f(n)})$ y hay que ejecutarla t veces. Para simular todas las ramificaciones de la máquina de Turing indeterminista, el número de pasos debe ser de orden exponencial $O(2^{f(n)})$. Por tanto, la complejidad temporal queda de orden exponencial sobre $f^2(n)$.
Pero, en cuanto a complejidad espacial cada ejecución necesita almacenar el valor de cm que ocupa un espacio de orden O(f(n)). Como la ejecución es recursiva, es necesario mantener una pila de configuraciones cm. La profundidad de la pila es de orden log(t), ya que en cada paso se divide t por la mitad. Pero log(t) es de orden $O(\log(2^{f(n)})$, es decir, de orden O(f(n)). Por tanto el espacio total es de orden O(f(n) * f(n)), es decir, $O(f^2(n))$.

Febrero 2020, Febrero 2019 y Febrero 2016
EJERCICIO 7 (1 punto)
(a) ¿Qué es un problema PSPACE?

Todo problema de la clase P (complejidad temporal polinomial) pertenece a la clase PSCPACE.

(b) ¿Qué es un problema NPSPACE?

Todo problema de la calse NP (complejidad temporal polinomial sobre máquinas no deterministas) pertenece a la clase NPSPACE.

(c) Enuncie el Teorema de Savitch.

Toda máquina de Turing indeterminista con una complejidad espacial de orden $f(n) \geq n$ puede convertirse en una máquina de Turing determinista con una complejidad espacial de orden $f^2(n)$.

Septiembre 2019 ejercicio 7
(c) ¿Qué es un problema PSPACE-completo?

Se dice que un lenguaje B es PSPACE-completo si cumple dos condiciones:
1. B pertenece a PSPACE
2. Cualquier problema A perteneciente a PSPACE se puede reducir a B en tiempo polinomial.