Parcial FTC - 2024

Ejercicio 1

Punto A

Un lenguajes es **recursivo** cuando cuenta con una *máquina de Turing M que acepte a ese lenguaje y pare para todas las instancias del problema ya sean positivas o negativas.*

Punto B

Lenguaje recursivo:

- $SAT = \{\varphi \mid \varphi \text{ es una fórmula booleana satisfactible con m variables en la FNC}\}$
  Lenguaje no recursivo:
- $L_{\Sigma*} = \{< M > \ \mid L(M) = \Sigma*\}$

Punto C

Si un lenguaje L es recursivo, su complemento L^C también será recursivo. Por definición L tiene una MT M que lo decide y para siempre, para probar que su complemento también cumple esa definición podemos crear una MT $M´$ que simule M e invierta las salidas de sus estados finales, de esta forma $M´$ parará siempre para toda instancia positiva o negativa correspondiente a L^C ya que M paraba en un primer lugar.

Ejercicio 2

Idea General de M

- Lo que la máquina M puede hacer para mostrar la intersección entre los dos lenguajes de $L_1 \ y \ L_2 \in RE$ sería que dada una cadena de entrada w M simulará secuencialmente $M_1 \ y \ M_2$ sobre esa cadena y aceptará w sii las 2 MT aceptan, en caso de que alguna rechace, M rechazará w, y en caso de que alguna de las 2 MT loopee, M también loopeará.

Ejercicio 3

Idea General de M_D

- Lo que la máquina M_D hará para reconocer D es:
  - Dada la entrada w_i de entrada tomará el índice de la cadena y generará códigos válidos de máquinas de Turing según el orden canónico hasta dar con la $i-ésima$

máquina de Turing según el orden canónico.
- Una vez encontrada la $i-ésima$ máquina de Turing según el orden canónico, M_D simulará la cadena w sobre la máquina M_i:
  - Si M_i acepta, M_D acepta.
  - Si M_i rechaza, M_D rechaza.
  - Si M_i loopea, M_D loopea.

Ejercicio 4

Se cumpliría que $RE = R$ si el lenguaje HP fuera fuera recursivo por que, ya que se cumple que para todo $L_i \in R$ existe una reducción de $L_i \leq L_{HP}$, por propiedad de las reducciones $L_1 \leq L_2$, entonces $L_2 \in R \rightarrow L_1 \in R$, estaríamos diciendo que todo lenguajes perteneciente a RE también sería recursivo.

Ejercicio 5

La función indicada no es una reducción de $L_U \leq L_U^C$ ya que, si bien cumple con la propiedad de ser **total computable** ya que está definida para todo el alfabeto del problema y para siempre, no cumple con la propiedad de **correctitud** esto se debe a que $L_U \in RE$ esto significa que posee una MT que acepta el lenguaje y para siempre para todas las instancias positivas del problema, para las instancias negativas puede parar y rechazar o quedarse loopeando, pueden ocurrir casos donde una cadena $w \notin L_U$ ya que M_U loopea y como la M^C funciona igual que la M_U pero solo invierte sus estados finales también lopee, algo que no sería correcto ya que por definición del complemento w debería de pertenecer a L_U^C.

Ejercicio 6

Punto A

Un lenguaje pertenece a la clase P cuando cuenta con una **máquina de Turing determinística que lo decide en tiempo polinomial con respecto a la entrada**, es decir, tiene orden $O(n^K)$ siendo K una constante.

Punto B

Lenguaje dentro de P

- El lenguaje de los *palíndromos* $L = \{w \mid \text{w es un palíndormo con símbolos a y b}\}$ está en P ya que la MT que lo decide lo hace en tiempo polinomial, copia la entrada en dos cintas y va recorriendo una cinta de izquierda a derecha, y la otra de derecha a izquierda, y por cada posición compara los símbolos que lee, esto se hace en tiempo polinomial.

Lenguajes fuera de P

- El lenguaje $SAT = \{\varphi \mid \varphi \text{ es una fórmula booleana satisfactible con m variables en la FNC}\}$ está por fuera de P ya que en el peor de los casos para una fórmula φ de entrada debemos analizar sus 2^N posibles asignaciones, siendo N la cantidad de variables de φ, esto hace que la solución se haga en un tiempo exponencial.

Punto 7

Punto A

Un lenguaje pertenece a la clase NP si cumple alguna de estas 2 posibles condiciones:

1. Si una cadena w pertenece al lenguaje, existe un certificado sucinto (de tamaño polinomial con respecto a la entrada) verificable en tiempo polinomial por una MT verificadora del lenguaje.
2. Se puede construir una máquina de Turing no determinística que decida al lenguaje en tiempo polinomial.

Punto B

Lenguaje dentro de NP

- El lenguaje $SAT = \{\varphi \mid \varphi \text{ es una fórmula booleana satisfactible con m variables en la FNC}\}$ es $NP - completo$. y por lo tanto, está en NP. Los certificados de sus cadenas serían una lista con la asignación de valores que debemos darles a las variables para que la fórmula φ sea satisfactible, estos certificados tiene tamaño N, que es polinomial con respecto a la entrada y se verifican en tiempo polinomial.

Ejercicio 8

Para que L_2 sea $NP - completo$ tiene que cumplir 2 condiciones:

- Pertenecer a NP.
- Ser $NP - dfícil$, es decir, para todo $L_i \in NP$ existe una reducción polinomial $L_i \leq_p L_2$.

La primera condición ya la cumple por enunciado. La segunda condición la cumple por propiedad de transitividad de las reducciones porque se nos dice que L_1 es $NP - completo$ por lo que se cumplen las condiciones de arriba, eso hace que para todo $L_i \in NP$ existe una reducción polinomial $L_i \leq_p L_1$, y también existe una reducción polinomial $L_1 \leq_p L_2$, por lo tanto, para todo $L_i \in NP$ existe una reducción polinomial $L_i \leq_p L_2$. *De esta forma se cumplen las dos condiciones, por lo tanto, L_2 es $NP - completo$.*

Ejercicio 9

Punto A

$TAUT$ no estaría en P por una justificación muy parecida a la de SAT, tenemos que comprobar las 2^N posibles asignaciones de la fórmula booleana para poder decidir que esa fórmula es una tautología.

Punto B

$TAUT$ no estaría en NP porque no posee certificado sucinto, la forma de su certificado tendría que ser una lista con todas las posibles asignaciones para las variables de la fórmula que la hacen satisfactoria, esto hace que el certificado posea tamaño exponencial.

Ejercicio 10

Punto A

En la jerarquía espacial consideramos lenguajes tratables a los lenguajes que pertenecen a la clase $LOGSPACE$ (poseen una MT que los deciden en espacio logarítmico), esto se debe a que estos lenguajes además de ocupar espacio logarítmico, se pueden decidir en un tiempo con orden $O(c^{log_2(n)}) = O(n^{log_2(c)})$, es decir, polinomial.

Punto B

Una MT que ocupa espacio $poly(n)$ no tiene por qué tardar tiempo $poly(n)$ debido a que el espacio que ocupa una MT no está directamente relacionado con el tiempo que tarda esa MT, una MT que trabaja en espacio $poly(n)$ se ve acotada en tiempo por la cantidad de configuraciones distintas por las que puede pasar antes de loopear para una cadena de entrada, en este caso esas configuraciones se calculan como $c^{S(n)}$ lo que es a lo sumo exponencial.

Punto C

Idea General

- Para imprimir una cadena con $S(n)$ marcas X, lo que hará M_2 es simular M_1 paso a paso y por cada paso que haga M_1 se va verificar si M_1 accedió a una nueva posición de la cinta, en tal caso, escribiremos una X en una cinta distinta, al final de simular M_1, como esta ocupa $S(n)$ celdas, M_2 habrá terminado de escribir una cadena de $S(n)$ marcas X.