Universidad Nacional de La Plata

Facultad de informática

Teoría de la computación y verificación de programas

Prácticas - Resolución

Autor:

Lautaro Castro

Importante: Las prácticas fueron entregadas y corregidas. Algunas correciones no fueron resueltas, sin embargo estas pueden verse en los siguientes enlaces:

- Práctica 1: https://docs.google.com/document/d/1m3k70pK-n__vUywpGTcsf2kl4jYrEQ_xfUbxQRnn_2Q/edit?usp=sharing
- Práctica 2: https://docs.google.com/document/d/197Z0uG4Ghi21SbLF2kQ8yfBD8xps-lJM/edit?usp=sharing&ouid=108608161596435259757&rtpof=true&sd=true
- Práctica 3: https://docs.google.com/document/d/1oXx9LoLuxprCJ4jhewfxAL1SG-luawfu/edit?usp=sharing&ouid=108608161596435259757&rtpof=true&sd=true

Que un ejercicio (en esos enlaces) tenga un comentario implica simplemente una aclaración o que el problema cuenta con un error, caso contrario el ejercicio esta correctamente resuelto.

Índice

1. Práctica 1 - Máquinas de Turing. Jerarquía de computabilidad

Comentario: Los ejercicios 5 y 6 son un poco más difíciles, resolverlos da un mayor bonus para la calificación de la materia.

1.1. Ejercicio 1

Enunciado: Responder breve y claramente los siguientes incisos:

Resolución

1. ¿Qué es un problema computacional de decisión? ¿Es el tipo de problema más general que se puede formular?

Un problema computacional de decisión es un problema donde a partir de un dato de entrada debe responderse por Si o por No de acuerdo al problema planteado. Por ejemplo, "dado un grafo G y dos vertices v1, v2 responder si existe un camino entre ellos".

No, no es el tipo de problema más general que se puede formular, están también los **problemas de búsqueda**, donde se resuelve un problema generando su solución. por ejemplo, "dado un grafo G y dos vértices v1, v2 computar el camino correspondiente".

2. Dados $\Sigma = \{a, b, c\}$ y $L = \{a^n b^n c^n | n \geq 0\}$, obtener $\Sigma^* \cap L$, $\Sigma^* \cup L$, y el complemento de L con respecto a Σ^*.

- $\Sigma^* \cap L = L$
  1. Por definición de Σ^* (conjunto de todos los símbolos formables a partir del alfabeto universal), sea l un lenguaje $(\forall l)(l : l \subseteq \Sigma^*)$
  2. Concluyendo en que $\Sigma^* \cap L = L$
- $\Sigma^* \cup L = \Sigma^*$
  1. Por definición de Σ^* (conjunto de todos los símbolos formables a partir del alfabeto universal), sea l un lenguaje $(\forall l)(l : l \subseteq \Sigma^*)$
  2. Concluyendo en que $\Sigma^* \cup L = \Sigma^*$
- $L^C = \Sigma^* - L$ (Por definición de complemento)

3. En la clase teórica 1 se hace referencia al problema de satisfactibilidad de las fórmulas booleanas. Formular las tres formas del problema, teniendo en cuenta las tres visiones de MT consideradas: calculadora, aceptadora o reconocedora, y generadora.

- **Problema de decisión (visión reconocedora):** Dada una fórmula booleana φ ¿Existe una asignación de valores A que la satisfaga?
- **Problema de búsqueda (visión calculadora):** Dada una formula booleana φ, encontrar la asignación de valores A que la satisfaga.
- **Problema de búsqueda (visión generadora):** Encontrar todas las fórmulas booleanas φ tal que existe una asignación de valores A que la satisfaga.

4. ¿Qué postula la Tesis de Church-Turing?

Postula que todo proceso computable es realizable por una Maquina de Turing.

5. ¿Cuándo dos MT son equivalentes? ¿Cuándo dos modelos de MT son equivalentes?

- Dadas dos MT M_1 y M_2, son equivalentes cuando $L(M_1) = L(M_2)$
- Sea M_A una MT del modelo A y M_B una MT del modelo B. Los modelos A y B son equivalentes si se comprueba que para todo:

- $L(M_A) = L(M_B)$
- $L(M_B) = L(M_A)$

6. ¿En qué difieren entre sí los lenguajes recursivos, los lenguajes recursivamente numerables no recursivos, y los lenguajes no recursivamente numerables?

- **Lenguajes recursivos:** Son los lenguajes que son reconocidos por una MT M que siempre se detiene para cualquier entrada.
- **Lenguajes recursivamente numerables no recursivos:** Son los lenguajes que son reconocidos por una MT M que no siempre se detienen para cualquier entrada.
- **Lenguajes no recursivamente numerables:** Son lenguajes que no son reconocidos por ninguna MT M.

7. Probar que $R \subseteq RE \subseteq \mathcal{L}$. Ayuda: usar directamente las definiciones.

Notar que tanto R, RE y $\mathcal{L}$ son conjuntos de conjuntos

1. Por definición $\mathcal{L} = P(\Sigma^*)$, entonces $\mathcal{L}$ es el conjunto de todos los lenguajes (los lenguajes son conjuntos) formables a partir del alfabeto universal.
2. Por la definición anterior entonces $RE \subseteq \mathcal{L}$. Además RE es el conjunto de los lenguajes reconocidos por una MT M (lo cual no asegura que siempre se detenga para cualquier entrada).
3. Por definición R es el conjunto de lenguajes reconocidos por una MT M que siempre se detiene, siendo R entonces un caso particular de RE, por lo tanto $R \subseteq RE$

8. ¿Qué lenguajes de la clase CO-RE tienen MT que los aceptan? ¿También los deciden?

Los lenguajes de CO-RE que tienen una MT que los aceptan están incluidos por definición en RE, y como es sabido que $R = CO\text{-}RE \cap RE$, este tipo de lenguajes son recursivos (es decir que son problemas computables decidibles). Esto puede comprenderse más claramente con la siguiente imagen:

9. Justificar por qué los lenguajes universal Σ^* y vacío $\varnothing$ son recursivos.

Nota: Con la idea general es suficiente, no es necesario la construcción de las MTs (eso fue hecho únicamente a modo de práctica)

Demostración $\Sigma^* \in R$

La idea general es la de una MT que se detiene en q_A para toda cadena de entrada en un solo paso.

1. **Construcción de la MT**

   Sea M = <Q, Σ, Γ, δ, q_0, q_A, q_R>

   $Q = \{q_0\}$

   $\Sigma = \{s_1, s_2, ..., s_n\}$

   $\Gamma = \Sigma \cup \{B\}$

   $\delta : Q \times \Gamma \to Q \cup \{q_A, q_R\} \times \Gamma \times \{L, R, S\}$

   - $\delta(q_0, x) = (q_A, x, S) \;\; (\forall x)(x : x \in \Sigma \cup \{B\})$

2. **Prueba de correctitud**

   - $w \in \Sigma^*$
     $\Rightarrow w \in L(M)$ (Por construcción de M, se acepta cualquier w en un solo paso)

- $w \notin \Sigma^*$ (Falso, por definición de Σ^*)

  $w \notin L(M)$ (Falso por construcción de M, este acepta cualquier w en un solo paso)

$\therefore L(M) = \Sigma^*$

Demostración $\varnothing \in R$

La idea general es la de una MT que se detiene en q_R para cualquier entrada en un solo paso.

1. **Construcción de la MT**

   Sea M = <Q, Σ, Γ, δ, q_0, q_A, q_R>

   $Q = \{q_0\}$

   $\Sigma = \{s_1, s_2, ..., s_n\}$

   $\Gamma = \Sigma \cup \{B\}$

   $\delta : Q \times \Gamma \to Q \cup \{q_A, q_R\} \times \Gamma \times \{L, R, S\}$

   - $\delta(q_0, x) = (q_R, x, S) \ \ (\forall x)(x : x \in \Sigma \cup \{B\})$

2. **Prueba de correctitud**

   - $w \in \varnothing$ (Falso, por definición de $\varnothing$)

     $w \in L(M)$ (Falso por contrucción de M, se rechaza cualquier w en un solo paso)

   - $w \notin \varnothing$

     $\Rightarrow w \notin L(M)$ (Por construcción de M, se rechaza cualquier w en un solo paso)

$\therefore L(M) = \varnothing$

10. Justificar por qué un lenguaje finito es recursivo.

Un lenguaje finito es recursivo porque existe una MT M que la reconoce y siempre se detiene para cualquier cadena de entrada. Esta funcionaría de la siguiente manera:

1. Se imprime en una cinta todas las cadenas del lenguaje. Al ser finito esto terminará eventualmente
2. Se compara la cadena de entrada con cada cadena generada en el paso (1).
   - Si alguna comparación es válida para en q_A
   - Si ninguna comparación es válida para en q_R

Como puede notarse la máquina siempre se detiene, la cantidad de pasos para generar el lenguaje es finita debido a que el lenguaje es finito, luego las comparaciones son procesos que también se realizan en un tiempo finito y finalmente ya sea porque una comparación es válida o ninguna comparación es válida se parará en q_A o q_R respectivamente.

Demostración formal (hecho de forma adicional a modo de práctica, la primer respuesta es suficiente y válida):

La demostración consta de dos partes. En la primera se demostrará que sea L un lenguaje tal que consta de una única palabra w, este pertenece a R. Luego a partir de este teorema se llegará facilmente a la conclusión de que dado un lenguaje finito este es recursivo.

Teorema 1: Si L es un lenguaje de una única palabra, entonces $L \in R$

1. **Idea general**
   Sea w la única palabra que acepta un lenguaje L, podemos decir que w es la concatenación de una serie de símbolos s_i, es decir $w = s_0.s_1.s_2. \ ... \ .s_n$. Entonces se puede construir una MT M donde cada estado q_i necesita del símbolo s_i para pasar a la siguiente transición hasta llegar a s_n que transiciona a q_A, y donde cualquier símbolo s_j tal que $j \neq i$, implica una transición hacia q_R en el estado q_i
2. **Construcción de la MT**
   $M =< Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R >$

   $Q = \{q_0, q_1, ..., q_n, \}$

   $\Sigma = \{x_0, x_1, ..., x_n\}$

   $\Gamma = \Sigma \cup \{B\}$

   $\delta = Q \times \Sigma \to Q \cup \{q_A, q_R\} \times \Sigma \times \{D, I, S\}$

   - $(q_i, s_i) = (q_{i+1}, s_i, D) \quad (\forall i)(i : i >= 0 \wedge i < n)$
   - $(q_n, s_n) = (q_A, s_n, D)$
   - $(q_i, s_j) = (q_R, s_j, D) \quad (\forall i)(i : 0 >= i <= n \wedge i \neq j)$
3. **Prueba de correctitud**

   a) $w \in L$

   $\Rightarrow w \in L(M)$ (Por construcción de M, en n+1 pasos (+1 por pasar de q_n a q_A) llega a q_A)

   b) $w \notin L$

   $\Rightarrow w \notin L(M)$ (Por construcción de M, si el símbolo en la posición i no coincide con el esperado en el estado q_i la cadena se rechaza)

Teorema 2: Si L es un lenguaje finito entonces $L \in R$

Se puede demostrar que la proposición es verdadera, a través de los siguientes razonamientos:

1. L es un lenguaje finito, entonces este puede definirse como $L = \{w_1, w_2, ..., w_n\}$
2. Si tomamos las n-palabras, por **Teorema 1** para cada w_i existe un lenguaje MT M_i tal que $L(M_i) = \{w_i\} = L_i \in R$.
3. Entonces L puede definirse como la unión de todos los lenguajes L_i enunciados en (2), es decir $L = L_1 \cup L_2 \cup ... \cup L_n$, y como R es cerrada respecto a la unión, puede concluirse que $L \in R$

11. Justificar por qué si $L_1 \in$ CO-RE y $L_2 \in$ CO-RE, entonces $(L_1 \cap L_2) \in$ CO-RE. Ayuda: una manera de resolverlo es utilizando las leyes de De Morgan del álgebra de Boole.

1) $L_1 \in CO\text{-}RE \land L_2 \in CO\text{-}RE$ (Por hipótesis)
2) $L_1^C \in RE \land L_2^C \in RE$ (Por definición de CO-RE)
3) $L_1^C \cup L_2^C \in RE$ (La unión es cerrada en RE)

   $\Rightarrow (L_1 \cap L_2)^C \in RE$ (Ley de Morgan)

   $\therefore (L_1 \cap L_2) \in CO\text{-}RE$

1.2. Ejercicio 2

Enunciado: Construir una MT, con cualquier cantidad de cintas, que acepte de la manera más eficiente posible el lenguaje $L = \{a^n b^n c^n | n \geq 0\}$. Comentario: Plantear primero la idea general.

Resolución

Idea general

La idea general es escanear la cinta de entrada para contar todas las "a". Para ello se utilizarán 2 cintas más donde por cada "a" se escribirá la misma cantidad de "b" en la cinta 2 y "c" en la cinta 3. Entonces, luego de escanear las "a", se empezará a comparar los simbolos "b" de la cinta de entrada contra la cinta 2, y luego los símbolos "c" contra la cinta 3. Si en algún caso la cantidad de símbolos no es la correcta entonces se rechaza.

Construcción de la MT

$M =< Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R >; \quad k = 3$

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{a, b, c\}$

$\Gamma = \Sigma \cup \{B\}$

$\delta : Q \times (\Gamma)^k \to Q \cup \{q_A, q_R\} \times \Gamma^k \times \{I, D, S\}^k$

- $\delta(q_0, (B, B, B)) = (q_A, (B, B, B), (S, S, S))$
- $\delta(q_0, (a, B, B)) = (q_1, (a, B, B), (S, S, S))$
- $\delta(q_1, (a, B, B)) = (q_1, (a, b, c), (D, D, D))$
- $\delta(q_1, (b, B, B)) = (q_2, (b, B, B), (S, I, I))$
- $\delta(q_2, (b, b, c)) = (q_2, (b, B, c), (D, I, S))$
- $\delta(q_2, (c, B, c)) = (q_3, (c, B, c), (S, S, S))$
- $\delta(q_3, (c, B, c)) = (q_3, (c, B, B), (D, S, I))$
- $\delta(q_3, (B, B, B)) = (q_A, (B, B, B), (S, S, S))$
- Para el resto de transiciones no definidas se para en q_R

Adicional

La ejecución de esta MT puede probarse en https://turingmachinesimulator.com: seleccione Examples→Binary Addition o Binary Multiplication, pegue el siguiente código fuente y **compile**:

```
name: L = a^n b^n c^n / n >= 0
init: q0
accept: qA

q0,_,_,_
qA,_,_,_,-,-,-

q0,a,_,_
q1,a,_,_,-,-,-

q1,a,_,_
q1,a,b,c,>,>,>

q1,b,_,_
q2,b,_,_,-,<,<

q2,b,b,c
q2,b,_,c,>,<,-

q2,c,_,c
q3,c,_,c,-,-,-

q3,c,_,c
q3,c,_,_,>,-,<

q3,_,_,_
qA,_,_,_,-,-,-
```

1.3. Ejercicio 3

Enunciado: Ejercicio 3. Explicar (informal pero claramente) cómo simular una MT por otra que en un paso no pueda simultáneamente modificar un símbolo y moverse.

Resolución

La simulación se puede realizar de la siguiente forma:

1. Todas las transiciones con movimiento S (static) se mantienen sin modificación.
2. Para las transiciones con movimiento R o D que escriben símbolos se simulan de la siguiente forma. Supongamos la transición-i que modifica un símbolo, se mueve y pasa al estado q', entonces se realizan los siguientes pasos:
   - Se ejecuta la transición, escribe el símbolo, pero no realiza el movimiento y pasa a un estado q_{Ri} o q_{Li} respectivamente
   - En este estado q_{Ri} o q_{Li} se mueve a R o D respectivamente y pasa al estado q'.

1.4. Ejercicio 4

Enunciado: Probar:

Resolución

1. La clase R es cerrada con respecto a la operación de unión. Ayuda: la prueba es similar a la desarrollada para la intersección.

Sean L_1 y L_2 dos lenguajes que pertenecen a R, y M_1, M_2 las respectivas máquinas que las reconocen y siempre se detienen:

Idea general

La unión es juntar los elementos de ambos en un solo conjunto. Entonces se podrían ejecutar tanto M_1 y M_2 de forma secuencial, así si la cadena de entrada es aceptada por alguna, implica que pertenece a la unión.

Construcción de la MT

Sea w la cadena de entrada, se puede construir una MT M' con el siguiente funcionamiento:

1. Ejecuta M_1 sobre w. Si la acepta entonces M' para en q_A, si se detiene en q_R entonces ejecuta el paso (2).
2. Ejecuta M_2 sobre w. Si la acepta entonces M' para en q_A, si M_2 se detiene en q_R entonces M' para en q_R

Prueba de correctitud

- $w \in L_1 \cup L_2 \Rightarrow w \in L(M')$ esto es correcto por construcción de M', si la cadena pertenece a L_1 o L_2 será reconocida por M_1 o M_2 y por lo tanto reconocida por M'.
- $w \notin L_2 \cup L_2 \Rightarrow w \notin L(M')$ esto es correcto por construcción de M', si la cadena no pertenece a L_1 o L_2, M_1 y M_2 pararán en q_R por lo cual M' también parará en q_R.

2. La clase RE es cerrada con respecto a la operación de intersección. Ayuda: la prueba es similar a la desarrollada para la clase R.

Sean L_1 y L_2 lenguajes que pertenecen a RE, y M_1, M_2 las respectivas máquinas que las reconocen:

Idea general

La intersección es el conjunto de las cadenas que están en ambos lenguajes, entonces la idea es simular la cadena de entrada sobre M_1 y M_2, si esta es aceptada por ambas entonces esta pertenece a su intersección. Para evitar posibles loops la ejecución debe realizarse concurrentemente de a pasos.

Construcción de la MT

Sea w la cadena de entrada, se puede construir una MT M' multicinta con el siguiente funcionamiento:

1. Se inicializa un contador de pasos p = 1.
2. Se ejecuta en una cinta M_1 a lo sumo p pasos sobre w.
3. Se ejecuta en una cinta M_2 a lo sumo p pasos sobre w.
4. Si ambas aceptaron w entonces M' para en q_A, si alguna se detuvo en q_R entonces M' rechaza a w. Caso contrario se ejecutan ambos (2) y (3) o solo uno (si alguna ya reconoció) respectivamente con p = p + 1.

Prueba de correctitud

- $w \in L_1 \cap L_2 \Rightarrow w \in L(M')$, por construcción de M' si w pertenece a la intersección entonces w será reconocida por M_1 y M_2 en una cantidad finita de pasos y M' se detendrá en q_A

- $w \notin L_1 \cap L_2 \Rightarrow w \notin L(M')$, por construcción de M' si w no pertenece a la intersección entonces w no será reconocido por M_1 y/o M_2, por lo cual M' tampoco reconocerá w.

1.5. Ejercicio 5

Enunciado: Sean L_1 y L_2 dos lenguajes recursivamente numerables de números naturales codificados en unario (por ejemplo, el número 5 se representa con 11111). Probar que también es recursivamente numerable el lenguaje $L = \{x \mid x$ es un número natural codificado en unario y existen y, z, tales que $y + z = x$, con $y \in L_1, z \in L_2\}$. Ayuda: la prueba es similar a la vista en clase, de la propiedad de clausura de la clase RE con respecto a la operación de concatenación.

Resolución

Idea general

La idea general es probar todas las formas posibles de concatenación (notar que la suma de dos cadenas unarias, es una concatenación) de la cadena de entrada w, emulando cada parte de la concatenación en la máquina correspondiente L_1 o L_2 de modo que si ambas aceptan su parte implica que la concatenación es válida. Así primero la cadena se divide en 0 símbolos del lado izquierdo y n del lado derecho, siendo |w| = n, luego 1 símbolo del lado izquierdo y n-1 del lado derecho, así sucesivamente hasta que el lado izquierdo toma n símbolos y el lado derecho 0. Para evitar loops la prueba de todas las combinaciones se realiza de a pasos, primero pruebo todas con 1 paso, luego con 2 y así sucesivamente.

Construcción de la MT

Sea M una MT con 6 cintas, w la cadena de entrada que se recibe en la cinta 1 donde |w| = n, L_1 tal que $L(M_1) = L_1$, L_2 tal que $L(M_2) = L_2$. Se describirá el funcionamiento de M en pasos:

1. Se escribe el valor de la cantidad de pasos a realizar en la cinta 2. Sea este valor p que inicialmente vale 1
2. Se escribe la cantidad de símbolos a tomar del lado izquierdo en la cinta 3. Sea este valor C_I que inicialmente vale 0
3. Se escribe la cantidad de símbolos a tomar del lado derecho en la cinta 4. Sea este valor C_D que inicialmente vale n
4. Se borra la cinta y se escribe los primeros C_I símbolos de w en la cinta 5. Y se simula M_1 a lo sumo p pasos.
5. Se borra la cinta y se escribe los últimos C_D símbolos de w en la cinta 6. Y se simula M_2 a lo sumo p pasos.
6. Llegado a este paso puede darse tres casos:
   - Ambas simulaciones llegaron a un estado q_A entonces $w \in L$ y M se detiene
   - una o ambas simulaciones no llegaron a q_A y $C_I \neq N$. Entonces Se repite el proceso desde el paso 2 con $C_I = C_I + 1, C_D = C_D - 1$
   - una o ambas simulaciones no llegaron a q_A y $C_I = N$. Entonces se repite el proceso desde el paso 1 con $p = p + 1$, y reseteando $C_I = 0, C_D = n$

Prueba de correctitud

a) $w \in L \Rightarrow w \in L(M)$, esto se cumple por construcción de M, ya que prueba todas las combinaciones de concatenación posibles a partir de w, por lo cual si w es la concatenación de dos cadenas w_1 y w_2 tal que $w_1.w_2 = w$ y $w_1 \in L_1 \ \wedge \ w_2 \in L_2$ entonces $w \in L(M)$

b) $w \notin L \Rightarrow w \notin L(M)$, por construcción de M si no hay una concatenación de dos cadenas w_1 y w_2 tal que $w_1.w_2 = w$ y $w_1 \in L_1 \ \wedge \ w_2 \in L_2$ entonces la máquina se quedará lopeando sin llegar a una resolución por lo cual se cumple que $w \notin L(M)$

1.6. Ejercicio 6

Enunciado: Dada una MT M1 con alfabeto $\Sigma = \{0, 1\}$:

Resolución

Ayuda para la parte (1): Si L(M1) tiene al menos una cadena, entonces existe al menos una cadena w de unos y ceros, de tamaño n, tal que M1 a partir de w acepta en k pasos. Teniendo en cuenta esto, pensar cómo M2 podría simular M1 considerando todas las cadenas de unos y ceros hasta encontrar eventualmente una que M1 acepte (¡cuidándose de los casos en que M1 entre en loop!)

1. Construir una MT M2 que determine si L(M1) tiene al menos una cadena.

Por practicidad definiré a $M_2 = \{\langle M \rangle : |L(M)| > 0\}$

Idea general

La idea general es analizar de forma concurrente por pasos cada instancia de Σ^* hasta el eventual encuentro de una instancia válida. Para llevarse a cabo, se necesitará una MT generadora M^* que genere Σ^* en orden canónico y separando cada cadena w con un símbolo especial, supongamos "#". Entonces se podría construir una MT M multicinta que en una cinta emule M^* de modo que cada vez que esta genera una nueva instancia (cuando coloque "#") esta ejecutará $\langle M \rangle$ p pasos sobre cada w generado hasta el momento por M^*, evitando quedar en loop y analizando todas las posibles entradas hasta el eventual encuentro de una instancia válida.

Construcción de la MT

Sea M_2 que recibe una MT $\langle M \rangle$ en la cinta 1, siendo una MT multicinta con 4 cintas. Se describirá paso a paso su procesamiento:

1. Se almacena en la cinta 2 la cantidad de pasos p cuyo valor inicial es 1.
2. Se emula sobre la cinta 3 M^* hasta que termine de generar una cadena (es decir cuando esta coloque un símbolo "#").
3. Se limpia la cinta 4 y se copia el primer string w generado en la cinta 3 por M^* luego se emula a lo sumo p pasos $\langle M \rangle$ sobre w. Si $\langle M \rangle$ no reconoció w se sigue con la siguiente cadena generada por M^*, y así sucesivamente.
    - Si $\langle M \rangle$ reconoce algún w, M_2 se detiene en q_A
    - Si $\langle M \rangle$ no reconoció ningún w, se vuelve nuevamente al paso 1 con $p = p + 1$.

Prueba de correctitud

a) $|L(M)| > 0 \Rightarrow \langle M \rangle \in M_2$, esto es válido debido a que M_2 recorre todos los w de Σ^* por lo cual si M reconoce alguna cadena, M_2 por construcción también lo reconoce y acepta a $\langle$M$\rangle$.

b) $|L(M)| = 0 \Rightarrow \langle M \rangle \notin M_2$, esta proposición es verdadera porque M_2 emula a M, si M no reconociera ninguna cadena M_2 tampoco reconocería ninguna.

2. ¿Se puede construir además una MT M_3 para determinar si $L(M_1)$ tiene a lo sumo una cadena? Justificar.

No, no puede construirse M_3. Esto debido a que la única manera de saber si el lenguaje que acepta M_1 tiene como máximo 1 cadena, es evaluando todas las cadenas de Σ^* y este es un conjunto infinito, por lo cual nunca podría terminar de evaluarse haciendo imposible la construcción de M_3.

2. Práctica 2 - Máquina de Turing universal. Indecibilidad. Reducciones.

Nota: Lo relacionado a la verificación de codificaciones de máquinas de turing válidas se considera un problema trivial, por lo cual en pruebas o demostraciones es un aspecto omitido para más simplicidad y evitar redundancias sobre este aspecto.

2.1. Ejercicio 1

Enunciado: Probar que el lenguaje $L_U = \{(\langle M \rangle, w) |$ M acepta w $\}$ pertenece a la clase RE.

Resolución

Se probará que $L_U \in RE$ a través de una reducción de $L_U \leq HP$, donde $HP = \{(\langle M \rangle, w) |$ M se detiene con entrada w$\}$

1. Definición de reducción

Sea $f(\langle M \rangle, w) = (\langle M' \rangle, w)$, donde M' es igual a M con la diferencia de que todos los estados q_R de M se cambian en M' por un estado q_{loop} cuyas transiciones loopean indefinidamente.

2. Computabilidad

Existe una MT M_f que computa f, reemplazando todos los q_R por un estado q_{loop} y que agrega tuplas q_{loop} que loopean indefinidamente.

3. Correctitud

- $(\langle M \rangle, w) \in L_U \rightarrow M$ acepta w $\rightarrow M'$ acepta w $\rightarrow (\langle M' \rangle, w) \in L_H P$
- $(\langle M \rangle, w) \notin L_U \rightarrow M$ loopea o rechaza w $\rightarrow M'$ loopea $\rightarrow (\langle M' \rangle, w) \notin L_H P$

2.2. Ejercicio 2

Enunciado: Justificar o responder según el caso:

a. Se puede decidir si una MT M, a partir de la cadena vacía λ, escribe alguna vez un símbolo no blanco. Ayuda: ¿Cuántos pasos puede hacer M antes de entrar en loop?

Sí, puede decidirse, debido a que M con la restricción de entrada λ tiene a lo sumo tantos pasos como estados, es decir |Q| pasos antes de entrar loop. Ya que cada estado de la MT tiene a lo sumo las siguientes opciones:

1. Escribe un símbolo
2. No escribe nada y se mantiene en su mismo estado
3. No escriba nada y pasa a otro estado

Entonces si realiza (1) se sabe que escribió algo, si realiza (2) entra en loop, y si cada estado realiza (3) luego de |Q| pasos la cinta tendría todos símbolos B y estaría en un loop ya que ningún estado cambió algún símbolo, repitiendo el ciclo.

Así de esta manera por cada paso de M se puede verificar si se escribió algo, de modo que si se alcanza los |Q| pasos sin escribir ningún símbolo va a implicar la repetición de un ciclo, ya sea por (2) o por que todas las máquinas realizaron (3).

b. Se puede decidir si a partir de una cadena w, una MT M que sólo se mueve a la derecha se detiene. Ayuda: ¿Cuántos pasos puede hacer M antes de entrar en loop?

Sí, puede decidirse, debido a que M con las restricciones descritas tiene a lo sumo $|Sigma| + |Q|$ pasos. Ya que cada entrada a lo sumo para analizar completamente el string usará $|\Sigma|$ pasos, ya luego como

solo se mueve a la derecha se encontrará con símbolos B donde a lo sumo se podrán realizar $|Q|$ pasos antes de entrar loop ya que cada transición tiene las siguientes opciones:

1. Se mantiene en el mismo estado
2. Cambia de estado

Entonces si realiza (1) entra en loop, y si se realiza (2) a lo sumo puede realizarse |Q| veces antes de entrar loop.

Así de esta manera si luego de $|\Sigma| + |Q|$ pasos no se llego a q_A o q_R implica un loop y se sabría que la máquina no se detiene con la entrada w.

c. Se puede decidir, dada una MT M, si existe una cadena w a partir de la cual M se detiene en a lo sumo 10 pasos. Ayuda: ¿Hasta qué tamaño de cadenas hay que chequear?

Sí, puede decidirse. La idea sería generar en orden canónico todas las cadenas hasta llegar a la 1er cadena con 11 símbolos, luego de eso simplemente sería chequear cada cadena a lo sumo 10 pasos.

d. Intuitivamente, ¿se puede decidir, dada una MT M, si existe una cadena w de a lo sumo 10 símbolos a partir de la cual M para?

No, no puede decidirse. Ya que la única forma de decidir es probando M, y si bien tendría forma de saber si reconoce alguna cadena (ejecución por pasos), no sería posible el reconocimiento negativo, ya que para ello debería probar toda cadena de 10 símbolos, lo cual en alguna MT podría implicar un bucle infinito.

2.3. Ejercicio 3

Enunciado: Considerando la reducción de HP a L_U descripta en clase, responder:

a. Explicar por qué la función identidad, es decir la función que a toda cadena le asigna la misma cadena, no es una reducción de HP a L_U.

Sea $f(\langle M\rangle, w) = f(\langle M\rangle, w)$, es decir la funcion identidad:

$$(\langle M\rangle, w) \in HP \rightarrow \text{ M acepta o rechaza w } \rightarrow \text{ M acepta w } \rightarrow (\langle M\rangle, w) \in L_U$$
$$\rightarrow \text{ M no acepta w } \rightarrow (\langle M\rangle, w) \notin L_U$$

$\therefore$ la función identidad no es una reducción de HP a L_U

b. Explicar por qué las MT M´ generadas en los pares $(\langle M'\rangle, w)$, o bien se detienen aceptando, o bien loopean.

Porque la M' es igual a M con la única diferencia de que todos sus símbolos q_R en la función de transición fueron reemplazados por q_A por lo cual la única forma de detenerse que tiene es aceptando.

c. Explicar por qué la función utilizada para reducir HP a L_U también sirve para reducir HP^C a L_U^C.

Sirve debido a que cuando una MT M pertenezca a HP^C, M va a loopear en consecuencia M' va a loopear y rechazará w por lo cual pertenece a L_U^C. En cambio cuando M no pertenezca a HP^C significa que M para ya sea reconociendo o rechazando en q_R, entonces M' va a aceptar w y va a pertenecer a L'_UC

- $(\langle M\rangle, w) \in HP^C \Rightarrow M$ loopea $\Rightarrow M'$ loopea $\Rightarrow (\langle M\rangle, w) \in L_U^C$
- $(\langle M\rangle, w) \notin HP^C \Rightarrow M$ se detiene en q_A o $q_R \Rightarrow M'$ para en $q_A \Rightarrow (\langle M'\rangle, w) \notin L_U^C$

Además en general, $L1 \leq L2$ si y solo si $L1^C \leq L2^C$. Utilizando la misma función de reducción.

Utilizando la misma función de reducción.

d. Explicar por qué la función utilizada para reducir HP a L_U no sirve para reducir L_U a HP.

$(\langle M \rangle, w) \notin L_U \Rightarrow M'$ para en q_A o loopea sobre w $\Rightarrow M'$loopea $\Rightarrow (\langle M' \rangle, w) \notin HP$
$\Rightarrow M'$para en $q_A \Rightarrow (\langle M' \rangle, w) \in HP$

$\therefore$ La función de HP a L_U no sirve para reducir de L_U a HP ya que si algo no pertenece a L_U esta función de reducción puede generar un M' que si pertenezca a HP.

e. Explicar por qué la siguiente MT Mf no computa una reducción de HP a L_U: dada una cadena válida (<M>, w), Mf ejecuta M sobre w, si M acepta entonces genera el par desalida (<M>, w), y si M rechaza entonces genera la cadena 1.

Porque Mf no es total computable debido a que si ejecuta una MT M que recibe es posible que esta quede en un bucle infinito y nunca termine de computar la salida.

2.4. Ejercicio 4

Enunciado: Sea el lenguaje $D_{HP} = \{w_i \mid$ Mi para desde wi, según el orden canónico$\}$. Encontrar una reducción de D_{HP} a HP.

Resolución:

1. Definición de reudcción

sea $f(w_i) = (\langle M_i \rangle, w_i)$, tal que M_i es la i-esima MT en orden canónico

2. Computabilidad

Existe una MT M_f que genera la i-esima M_i de la siguiente manera:

1. inicia un contador j = 0
2. genera de acuerdo al orden canónico la cadena w_j
3. Compara la cadena generada con w_i
   - Si es igual a w_j entonces genera en orden canónico las MT hasta llegar a M_j de acuerdo al orden canónico. Luego codifica como entrada a w_i.
   - Si es diferente incrementa j = j + 1. Y vuelve al paso (2).

3. Correctitud

- $w_i \in D_{HP} \Rightarrow M_i$ se detiene con $w_i \Rightarrow (\langle M_i \rangle, w_i) \in HP$
- $w_i \notin D_{HP} \Rightarrow M_i$ no para con la entrada $w_i \Rightarrow (\langle M_i \rangle, w_i) \notin HP$

2.5. Ejercicio 5

Enunciado: Sean TAUT y NOSAT los lenguajes de las fórmulas booleanas sin cuantificadores, respectivamente, tautológicas (satisfactibles por todas las asignaciones de valores de verdad), e insatisfactibles (ninguna asignación de valores de verdad las satisface). Encontrar una reducción de TAUT a NOSAT.

Resolución:

1. Definición de reducción

sea $f(\varphi) = (\neg\varphi)$, donde $\neg\varphi$ es simplemente la negación de φ

2. Computabilidad

Existe una MT M_f que lo único que hace es agregar la negación de forma codificada a la entrada φ

3. Correctitud

- $\varphi \in TAUT \Rightarrow \neg\varphi$ es una contradicción $\Rightarrow \neg\varphi \in NOSAT$
- $\varphi \notin TAUT \Rightarrow \neg\varphi$ tiene alguna asignación que es V $\Rightarrow \neg\varphi \notin NOSAT$

2.6. Ejercicio 6

Enunciado: Un autómata linealmente acotado (ALA) es una MT con una sola cinta, con la restricción de que su cabezal sólo puede moverse a lo largo de las celdas ocupadas por la cadena de entrada. Probar que el lenguaje aceptado por un ALA es recursivo. Ayuda: ¿en cuántos pasos se puede detectar que el ALA entra en loop?

Resolución:

1. Idea general

Sea w la cadena entrada y M_L el ALA que reconoce un lenguaje L, entonces la idea es ejecutar M_L, entonces si $w \in L$ se sabrá que M_L se detiene, pero si $w \notin L$ y no se detiene entonces podremos saber que está en loop debido a que a lo sumo podrá realizar $|w| * |Q| * \Gamma^{|w|}$ pasos antes de repetir una configuración, ya que por cada celda (|w| porque está acotada al tamaño de la cadena de entrada) podrá estar en algún estado q (|Q|) y con tantas combinaciones de símbolos en las celdas igual a $\Gamma^{|w|}$

2. Construcción de la MT

Se puede construir una MT M con el siguiente funcionamiento:

1. Se inicializa i = 0
2. Se ejecuta w sobre M_L, y se incrementa i por cada paso
   - Si M_L acepta w entonces M acepta.
   - Si M_L para en q_R entonces M para en q_R.
   - Si se alcanza la situación en que i = $|w| * |Q| * \Gamma^{|w|}$, entonces implica que M' está en un bucle y M rechaza

3. Prueba de correctitud

- $w \in L \Rightarrow M_L$ acepta w $\Rightarrow w \in L(M)$
- $w \notin L \Rightarrow w \notin L(M) \wedge M$ se detiene , esto es verdadero por construcción de M, ya que si M_L para en q_R M también lo hará y si M_L loopea cuando alcance los $|w| * |Q| * \Gamma^{|w|}$ pasos M reconocerá que está en un loop y parará en q_R.

$\therefore$ un lenguaje L reconocido por un ALA pertenece a R

2.7. Ejercicio 7

Enunciado: Construir una MT que genere todos los índices i tales que $(\langle M_i \rangle, w_i) \in HP$, según el orden canónico.

Resolución:

Hay dos respuestas posibles de acuerdo a como se interprete la pregunta

Interpretación 1: La MT simplemente debe generar los pares $(M_i, w_i) \in HP$

1. Idea general

La idea es simplemente generar los pares (M_i, w_i) de acuerdo al orden canónico y realizar una ejecución por pasos para evitar loops y encontrar los pares que pertenezcan a HP.

2. Construcción de la MT

Se puede construir una MT con el siguiente funcionamiento:

1. Se inicializa i = 0
2. Se genera la MT M_i de acuerdo al orden canónico.
3. Se genera la cadena w_i de acuerdo al orden canónico.
4. Se ejecutan todos los pares (M_i, w_i) a lo sumo i pasos
   - Si M_i se detuvo con w_i se lo considera que pertenece a HP y se lo pone en una cinta (dando por hecho que se están utilizando varias cintas) denotando que pertenece a HP.
5. Se incrementa i = i + 1, y se vuelve al paso (2).

Interpretación 2: La MT debe generar los pares $(M_i, w_i) \in HP$ en estricto orden canónico

No es posible la construcción de esa hipotética MT M. Para generar cada $(M_i, w_i) \in HP$ en orden canónico, antes debo verificar que todas los pares anteriores a (M_i, w_i), es decir los (M_j, w_j) que estén antes de (M_i, w_i) deberían ejecutarse y verificar que o pertenecen a HP o no pertenecen, y eso no es posible ya que la ejecución de estos podría ser un loop evitando la generación en orden canónico. Si no se pidiera generar los indices i en orden canónico la máquina podría construirse haciendo una ejecución por pasos y generando los que se vallan descubriendo.

3. Práctica 3 - Complejidad computacional

Comentario: Los ejercicios 4, 8 y 9 llevan bonus.

3.1. Prefacio

Debido a que las fronteras de las jerarquías se decide de acuerdo a si la función en tiempo de un algoritmo es polinomial, resulta importante tener en cuenta algunas propiedades. Sea R[x] el conjunto de los polinomios en función de una variable x con coeficientes en R, y $P(x), Q(x) \in R[x]$

1. $P(x) + Q(x) \in R[x]$ Cerradura bajo suma
2. $P(x) * Q(x) \in R[x]$ Cerradura bajo producto

3.2. Conceptos teóricos

3.2.1. Ejercicio 1

Responder las siguientes preguntas conceptuales:

a) El lenguaje de los palíndromos se puede decidir con una MT de una cinta y también con una MT con varias cintas, en tiempo $O(n^2)$ y $O(n)$, respectivamente. ¿Por qué es indistinta la cantidad de cintas utilizadas, considerando la jerarquía temporal que definimos?

A nivel de complejidad temporal se considera estándar o razonable el modelo de las MTD multicinta, porque cualquier problema que se resuelve en tiempo polinomial con K_i cintas, también puede resolverse en tiempo polinomial con K_j cintas. Esto debido a que en el peor de los casos, es decir, reducir de K cintas a 1 cinta, tiene retardo cuadrático.

Si con $K > 1$ cintas se tarda $poly_1(n)$, con 1 cinta a lo suma se tarda:

$$poly_1(n) + n^2 = poly_1(n) + poly_2(n) = poly(n) \quad \text{(cerradura bajo la suma de polinomios)}$$

b) Vimos que un algoritmo natural para encontrar un divisor que termine en 3 de un número N tarda $O(N)$ pasos. ¿Esto significa que el problema está en P?

Contexto del problema

Sea DIV-3 = {N | N es un número natural que tiene un divisor que termina en 3}. Teniendo en cuenta lo siguiente:

- M ejecuta a lo sumo N/10 iteraciones, es decir $O(N)$
- Cada división se puede hacer en tiempo $O(n^2)$
- M ejecuta $O(N \cdot n^2)$

Si se expresa N en términos de n:

- Con base unaria $n = |N| = N$, es decir O(n) lineal. Entonces M tarda tiempo $O(n \cdot n^2) = O(n^3) = poly(n)$
- Con base binaria $n = O(\log_2(N))$, entonces N = $O(2^n)$, así M tarda tiempo $O(2^n \cdot n) = exp(n)$
- Si N se codifica con cualquier otra base, tarda exp(n)

Respuesta

A nivel de complejidad temporal, se considera estándar o razonable representar los números con cualquier notación **no unaria**, debido a su inviabilidad práctica y a las inconsistencias que genera, ya que la única solución polinomial es la unaria mientras que con el resto de bases la solución es siempre exponencial.

Por los motivos nombrados anteriormente, el algoritmo polinomial con base unaria no se considera razonable, por lo cual DIV-3 no estaría en P.

c) Probar que si $T1(n) = O(T2(n))$, entonces TIME($T1(n)$) $\subseteq$ TIME($T2(n)$).

Definiciones

- $T_a(n) = O(T_b(n)) \Rightarrow T_a(n) \leq c.T_b(n), c > 0$
- $L \in TIME(t(n)) \Rightarrow$ Existe una MT que lo decide en tiempo $O(t(n))$

Resolución

Sea un lenguaje $L_1 \in TIME(T_1(n))$ con función $T_L(n))$:

1. $T_1(n) = O(T_2(n))$

   $\Rightarrow T_1(n) \leq c.T_2(n), c > 0$ (Definición de orden)

2. $L_1 = O(T_1(n))$ (Definición de TIME(t(n)))

   $\Rightarrow T_L(n) < c.T_1, c > 0$ (Definición de orden)

   $\Rightarrow T_L(n) < c.T_2(n), c > 0$ (Por (1), transitividad operadores de igualdad/desigualdad)

   $\Rightarrow L_1 = O(T_2(n))$

   $\Rightarrow L_1 \in TIME(T_2(n))$

$\therefore TIME(T_1(n)) \subseteq TIME(T_2(n))$

d) Probar que la asunción $NP \neq CO\text{-}NP$ es más fuerte que la asunción $P \neq NP$, es decir que la implica.

Debido a que P es cerrado respecto al complemento, si se demostrara que NP no lo es, estas clases no serían iguales ya que tendrían diferentes propiedades lo que implicaría $P \neq NP$.

$NP \neq CO\text{-}NP \ \wedge \ P = NP$

$\Rightarrow (\exists L)(L : L \in (NP - CO\text{-}NP))$

$\Rightarrow L \in P$ (Por asunción P = NP)

$\Rightarrow L^C \in P$ (P es cerrado respecto al complemento)

$\Rightarrow L^C \in NP$ (Por asunción P = NP)

$\Rightarrow L \in CO\text{-}NP \ \wedge \ L \notin CO\text{-}NP$ (Absurdo)

$\therefore NP \neq CO\text{-}NP$ implicaría $P \neq NP$

e) ¿Qué significa que si $L1 \leq_P L2$, entonces $L2$ es tan o más difícil que $L1$, en el marco de la complejidad temporal?

Significa que L_2 es tan o más costoso a nivel de tiempo de ejecución respecto a L_1

f) Mostrar certificados para los siguientes lenguajes, e indicar cuáles son sucintos: CH (el lenguaje de los grafos con un Circuito de Hamilton), SAT (el lenguaje de las fórmulas booleanas satisfactibles), NOSAT (el lenguaje de las fórmulas booleanas insatisfactibles), ISO (el lenguaje de los grafos isomorfos), NOCLIQUE (el lenguaje de los grafos que no tienen un clique de tamaño K).

- $SAT = \{\varphi \mid \varphi$ es una formula booleana sin cuantificadores, con m variables y satisfactible$\}$
  - Certificado: Una asignación de variables que haga la formula satisfactible
  - Es sucinto: Se deberá analizar una asignación de a lo sumo m variables para su verificación.
- $CH = \{G \mid G$ es un grafo no dirigido y tiene un circuito de Hamilton$\}$
  - Certificado: El camino que representa el circuito de Hamilton
  - Es sucinto: Se deberá recorrer el camino de a lo sumo V vertices y E arcos para su verificación.
- $NOSAT = \{\varphi \mid \varphi$ es una formula booleana sin cuantificadores, con m variables y no es satisfactible$\}$
  - Certificado: Todas las 2^m asignaciones de la fórmula booleana.
  - No es sucinto: Se deberán analizar las 2^m asignaciones para llevar a cabo la verificación.
- $ISO = \{(G_1, G_2) \mid G_1$ yG_2 son grafos isomorfos salvo por el nombre de sus vértices$\}$
  - Certificado: Una combinación de vértices entre G_1 y G_2 tal que G_2 tenga la misma forma que G_1 o viceversa.
  - Es sucinto: Se deberá analizar una permutación de tamaño V y validar a si los arcos en cada lado, es decir analizar a lo sumo E*2 arcos para ver si son idénticos.

g) Mostramos en clase una reducción polinomial del lenguaje CH (del problema del circuito hamiltoniano) al lenguaje TSP (del problema del viajante de comercio). En base a esto justificar:

i. Como TSP es NP-completo, entonces CH $\in$ NP.

- $TSP \in NPC$

  $\Rightarrow TSP \in NP$ (Definición de NPC)

  $\Rightarrow CH \in NP$ (Teorema $(L_1 \leq_P L_2 \ \wedge \ L_2 \in NP) \rightarrow L_1 \in NP$, por hipótesis $CH \leq_P TSP$)

ii. Como CH es NP-completo, entonces TSP es NP-difícil.

- $CH \in NPC$

  $\Rightarrow (\forall L)(L : L \in NP \ \rightarrow \ L \leq_P CH)$ (Definición de NPC)

  $\Rightarrow (\forall L)(L : L \in NP \ \rightarrow \ L \leq_P TSP)$ (Transitividad en reducciones polinomiales, por hipótesis $CH \leq_P TSP$)

  $\therefore TSP \in NPH$

h) ¿Por qué si un lenguaje es NP-completo, en la práctica se considera que no pertenece a la clase P?

Porque al intuirse a nivel práctico que $P \neq NP$, sería imposible que un lenguaje de NPC perteneciera a P, ya que si se encontrara, se demostraría que $P = NP$.

Lo cual formalmente puede verse de dos formas:

1) **Si existe un lenguaje L ∈ NPC ∧ L ∈ P**

   (1,1) $L \in NPC \ \wedge \ L \in P$

   (1,2) $\Rightarrow (\forall L')(L' : L' \in NP \ \rightarrow \ L' \leq_P L)$ (Definición de NPC)

   (1,3) $\Rightarrow (\forall L')(L' : L' \in NP \ \rightarrow L' \in P)$ (Teo $L_1 \leq_P L_2 \ \wedge \ L_2 \in P \rightarrow L_1 \in P$, por 1.1 y 1.2)

   (1,4) $\Rightarrow P = NP$

2) **Si P ≠ NP, NPC no podría estar en P**

   $L \in NPC$

   $\Rightarrow (\forall L')(L' : L' \in NP \ \rightarrow \ L' \leq_P L)$ (Definición de NPC)

   $\Rightarrow (\exists L_{NP})(L_{NP} : L_{NP} \in (NP - P) \ \wedge \ L_{NP} \leq_P L)$ (Por $P \subseteq NP$, y asunción de $P \neq NP$)

   $\Rightarrow L \notin P$ (Por teorema $L_1 \leq_P L_2 \ \wedge \ L_1 \notin P \rightarrow L_2 \notin P$)

i) Explicar cómo agregaría un lenguaje a la clase NPC a partir del lenguaje CLIQUE, que se sabe que está en la clase NPC.

A partir de conocer un lenguaje que pertenezca a la clase NPC, le técnica más simple para seguir agregando lenguajes, es a través de la reducción polinomial, encontrando una reducción de CLIQUE a un nuevo lenguaje L, lo cual es correcto por la propiedad de transitividad en las reducciones polinomiales.

De manera informal, si $(L_1 \in NP) \leq_P (L_2 \in NPC) \leq_P L_3$, por transitividad todos los lenguajes de NP, es decir los L_1 se reducen a L_3, por lo cual si L_3 perteneciera a NP sería NP-completo.

j) Probar que si L1 ∈ NPC y L2 ∈ NPC, entonces L1 $\leq_P$ L2 y L2 $\leq_P$ L1.

- $L_1 \in NPC$

  $\Rightarrow (\forall L')(L' : L' \in NP \ \rightarrow \ L' \leq_P L_1)$ (Definición de NPC)

  $\Rightarrow L_2 \leq_P L_1$ (Por definición de NPC, L_2 pertenece a NP)

- $L_2 \in NPC$

  $\Rightarrow (\forall L')(L' : L' \in NP \ \rightarrow \ L' \leq_P L_2)$ (Definición de NPC)

  $\Rightarrow L_1 \leq_P L_2$ (Por definición de NPC, L_1 pertenece a NP)

k) Justificar por qué se consideran tratables sólo a los problemas que se resuelven en espacio logarítmico.

Contexto

En las definiciones de Máquina de Turing que trabaja en tiempo de S(n) no se define explícitamente que las MT se detienen siempre, sin embargo, para cada MT M que ocupa espacio S(n) que **puede no parar**, existe una MT M' equivalente que ocupa espacio S(n) y se **detiene siempre**. Por ejemplo:

- una MT con 1 cinta de entrada de sólo lectura y 1 cinta de trabajo entra en loop luego de: $(n+2).S(n).|Q|.|\Sigma|^{S(n)} = O(c^{S(n)})$ pasos.

Respuesta

Como espacio S(n) implica tiempo $O(c^{S(n)})$. Entonces una MT que ocupa espacio $\log_a n$, tarda tiempo $O(c^{log_a n})$ y como $c^{log_a n} = n^{log_a c} = poly(n)$, implicando que $LOGSPACE \subseteq P$, razón por la cual se consideran **tratables en espacio** a los problemas en LOGSPACE.

l) Explicar por qué en el caso de que un lenguaje requiera para ser decidido mínimamente espacio lineal, se puede utilizar una MT con una cinta de entrada de lectura y escritura.

Porque de considerarse la cinta de entrada en el análisis temporal, de mínima todos los problemas se acotarían a una función lineal debido al tamaño mínimo que necesita la entrada. Y como se asume un lenguaje que de mínima se decide en espacio lineal, tomar en cuenta la cinta de entrada no afecta al análisis.

m) Justificar por qué una MT que se ejecuta en tiempo poly(n) ocupa espacio poly(n), y por qué una MT que ocupa espacio poly(n) puede llegar a ejecutar exp(n) pasos.

- Si una MT M se ejecuta en tiempo poly(n), realizará poly(n) pasos por lo cual a lo sumo podrá ocupar poly(n) celdas.
- De acuerdo a lo visto en el inciso (1-k), espacio S(n) implica tiempo $O(c^{S(n)}$, entonces si una MT M ocupara espacio poly(n), tardaría $O(c^{poly(n))}) = EXP(n)$ y de la misma manera si ocupara espacio exp(n), razón por lo cual no se consideran tratables polinomial ni exponencial.

3.3. P vs NP

3.3.1. Ejercicio 2

Sea el lenguaje SMALL-SAT = {φ | φ es una fórmula booleana sin cuantificadores en forma normal conjuntiva (o FNC), y existe una asignación de valores de verdad que la satisface en la que hay a lo sumo 3 variables con valor de verdad verdadero}. Probar que SMALL-SAT $\in$ P. Comentario: φ está en FNC si es una conjunción de disyunciones de variables o variables negadas, como p.ej. $(x1 \cup x2) \cap x4 \cap (\not x3 \cup x5 \cup x6)$.

3.3.2. Ejercicio 3

El problema del conjunto dominante de un grafo se representa por el lenguaje DOM-SET = {(G, K) | G es un grafo que tiene un conjunto dominante de K vértices}. Un subconjunto de vértices de un grafo G es un conjunto dominante de G, si todo otro vértice de G es adyacente a algún vértice de dicho subconjunto. Probar que DOM-SET $\in$ NP. ¿Se cumple que DOM-SET $\in$ P? ¿Se cumple que DOM-SETC $\in$ NP?

Idea general

La MT para decidir si G tiene un conjunto dominante de K vértices, consiste en analizar 1 a 1 todas las combinaciones posibles de vértices para ver si se forma un subconjunto dominante. Entonces existen $C = \binom{m}{k}$ secuencias de K vértices en V, siendo m la cantidad de vértices de V.

Análisis temporal

- Dado que se deben recorrer C combinaciones, entonces:

  $$DOM\text{-}SET = O(\binom{m}{k}) = O(m^m) = O(n^n) = exp(n)$$

  Razón por la cual **DOM-SET no estaría en P**

- Dado que el certificado sería una secuencia C de K vértices, la **verificación** involucraría chequear que todos los arcos de los vértices de C alcanzan todos los vértices de G, es decir, ir contando qué vértice alcanza cada arco de los K vertices de C:

  $$O(|E| \cdot |V|) = O(|G|^2) = O(n^2) = poly(n)$$

  Por lo cual DOM-SET está en NP.

3.3.3. Ejercicio 4

Probar que el lenguaje FACT = {(N, M1, M2) | N tiene un divisor primo en el intervalo [M1, M2]} está tanto en NP como en CO-NP.

Ayuda: Todo número natural N se descompone de una única manera en factores primos, los cuales concatenados no ocupan más de poly(\|N\|) símbolos

Este problema consiste en encontrar si un número N tiene un divisor primo en el intervalo [M1,M2], entonces un certificado válido sería su descomposición en factores primos. Donde en esta descomposición, el peor caso es el de un número potencia de 2, porque al ser el número 2, el primo más chico, este deberá repetirse tantas o más veces que cualquier otro número para formar el número correspondiente.

- Así, el certificado será una lista de $n = log_2(N)$ y $N = O(n) = poly(n)$, por lo cual el certificado **es sucinto**.
- De esta manera la **verificación** se realiza en tiempo poly(n), ya que un número N consiste en a lo sumo $log_2(N)$ factores primos, que deben recorrerse para verificar si hay alguno entre $[M1, M2]$, lo que tarda: $O(log_2(N)) = O(n) = poly(n)$. Entonces FACT está en NP.
- De manera análoga, $FACT^C$ tarda poly(n), ya que la verificación es igual que en FACT, con la única diferencia que en vez de verificar si hay algún factor entre $[M1, M2]$, verifica que no haya ningún factor entre ese rango. Por lo cual FACT también está en CO-NP.

7. Duplicar un nodo.

8. Cada subgrafo completo es único

3.4. Reducciones Polinomiales

3.4.1. Ejercicio 5

Mostrar que las reducciones polinomiales son reflexivas y transitivas.

a) Reflexividad: Una relación tiene la propiedad reflexiva, si todo elemento está relacionado consigo mismo. En este caso signfica $L \leq_P L$.

1. Función de reducción

Sea L un lenguaje cualquiera, si lo queremos reducir polinomialmente consigo mismo, la entrada no debe modificarse, es decir que debe aplicarse la función identidad. Entonces sea w la cadena de entrada, se define f(w) = w, tal que f es la función identidad

2. Computabilidad polinomial

Existe una MT M_f que en un único paso se detiene (claramente es polinomial) y deja la cadena de entrada tal cual como está, sin modificarla.

3. Correctitud

Sea $M_f(w)$ la salida que computa M_f:

- $w \in L \Rightarrow M_f(w) \in L$ (Esto se cumple porque $M_f(w) = w$)
- $w \notin L \Rightarrow M_f(w) \notin L$ (Esto se cumple porque $M_f(w) = w$)

b) Transitividad: Una relación tiene la propiedad transitiva, cuando, dado elementos a, b, c, si a está relacionado con b y b está relacionado con c, entonces a está relacionado con c. En este caso significa que $(a \leq_P b \ \wedge \ b \leq_P c) \Rightarrow a \leq_P c$

1. Función de reducción

Sea A, B, C lenguajes tales que: $A \leq_P B \ \wedge \ B \leq_P C$, donde f(x) es la función que reduce polinomialmente y g(x) para B y C. Para demostrar la transitividad, se deberá encontrar una función tal

que permita $A \leq_P C$ dadas las condicionaes ya nombradas. Entonces se define la función compuesta g(f(x)), que mapea primero de A a B, y luego de B a C, permitiendo la reducción de A a C.

2. Computabilidad polinomial

Sea M_g y M_f las MT que computan f(x) y g(x) y w la cadena de entrada, existe una MT M'_f que computa en tiempo poly(n) g(f(w)) = w', siendo w' un elemento perteneciente a C, donde M'_f emula primero M_f con entrada w y luego emula M_g con entrada $M_f(w)$:

1.1) Sabemos que f(x) al ser la función de la reducción polinomial de A a B se resuelve en tiempo polinomial, entonces emular M_f con entrada w, retorna una cadena de a lo sumo de tamaño poly(w), porque f(x) al ser polinomial a lo sumo podrá recorrer poly(w) celdas.

1.2) Como la salida de $M'_f(w)$ tiene tamaño poly(n), entonces al emular M_g con entrada poly(w), a lo sumo se realizan poly(n) pasos sobre una cadena de tamaño poly(w), lo cual se realiza en tiempo poly(n).

1.3) Entonces el tiempo final de $M'f$ es:

$$M_f(w) + M_g(M_f(w)) = poly(n) + poly(poly(n)) = poly(n) + poly(n) = poly(n)$$

3. Correctitud

- $w \in A \Rightarrow M_f(w) \in B \Rightarrow M_g(M_f(w)) \in C \Rightarrow M'_f(w) \in C$
- $W \notin A \Rightarrow M_f(w) \notin B \Rightarrow M_g(M_F(w)) \notin C \Rightarrow M'_f(w) \notin C$

3.4.2. Ejercicio 6

Asumiendo que Σ^* sólo tiene cadenas de unos y ceros de cero o más símbolos, se define, dada la cadena w, que $E(w)$ es su cadena espejo, obtenida reemplazando en w los unos por ceros y los ceros por unos (p.ej. $E(1001) = 0110$ y $E(\lambda) = \lambda$). Y se define que L es un lenguaje espejo si para toda cadena w distinta de λ cumple $w \in L \Leftrightarrow E(w) \in L^C$. Sea f la función que asigna a toda cadena w su cadena espejo $E(w)$. Responder:

a) ¿Es f una función total computable?

Definición: Definición. Una función $f : A \to B$ es total computable sii existe una MT M_f que a partir de cualquier $a \in A$, se detiene y computa f(a) en su cinta de output.

Se puede construir una MT M_f que compute E(w) para cada cadena w, esta MT consistiría únicamente en recorrer cada símbolo de w y cambiar los 1 por 0 y 0 por 1 respectivamente. De esta manera como Σ (cadena de entrada) es finita, M_f eventualmente terminará de recorrerla, se detendrá y computara E(w), por lo cual es total computable.

b) ¿Cuánto tarda una MT que computa f?

Dado que la M_f definida en (a), recorre cada símbolo de w una única vez, esta realiza |w| pasos, lo cual puede acotarse a O(n), por lo cual M_f trabaja en tiempo polinomial.

c) Si L es un lenguaje espejo, ¿se cumple que f es una reducción polinomial de L a L^C?

- $w \in L \Rightarrow E(w) \in L^C \Rightarrow M_f(w) \in L^C$
- $w \notin L \Rightarrow E(w) \notin L^C \Rightarrow M_f(w) \notin L^C$

$\therefore$ M_f es una reducción de L a L^C, y por lo visto en los incisos (a) y (b), es además una reducción polinomial.

3.5. Lenguajes NP-completos

3.5.1. Ejercicio 7

Probar:

a) Si los lenguajes A y B son tales que $A \neq \emptyset$, $A \neq \Sigma^*$ y $B \in P$, entonces $(A \cap B) \leq_P A$.

Dado que en una conjunción un elemento debe pertenecer a ambos, con la reducción de

b) Si $L_1 \leq_P L_2$, $L_2 \leq_P L_1$, y $L_1 \in NPC$, entonces $L_2 \in NPC$.

Definición: Sea L un lenguaje, para pertenercer a NPC debe cumplir dos condiciones:

1. $L \in NP$
2. $(\forall L')(L' : L' \in NP \rightarrow L' \leq_P L)$

Para demostrar que L_2 pertenece a NPC, deberemos probar las condiciones nombradas anteriormente:

1. $L_2 \leq_P L_1 \ \wedge \ L_1 \in NP$ (Hipótesis inicial y definición de NPC)

   $\Rightarrow L_2 \in NP$ (Teorema $(L \leq_P L' \ \wedge \ L' \in NP) \rightarrow L \in NP)$)
2. $L_1 \leq_P L_2$

   $\Rightarrow (\forall L')(L' : L' \in NP \rightarrow L' \leq_P L_1)$ (Definición de NPC)

   $\Rightarrow (\forall L')(L' : L' \in NP \rightarrow L' \leq_P L_2)$ (Transitividad en reducciones polinomiales)

$\therefore L_2 \in NPC$

c) Si un lenguaje es NP-completo, entonces su complemento es CO-NP-completo, es decir, está en CO-NP y todos los lenguajes de CO-NPC se reducen polinomialmente a él.

$L \in NPC \ \wedge \ L^C \in CO\text{-}NPC$

$\Rightarrow (\forall L')(L' : L' \in NP \rightarrow L' \leq_P L)$ (Definición de NPC)

$\Rightarrow (\forall L')(L' : L' \in NP \rightarrow L'^C \leq_P L^C)$ (Sean L_1 y L_2 tales que $L_1 \leq_P L_2$, la misma función de reducción sirve para los complementos, entonces $L_1^C \leq_P L_2^C$)

$\therefore L^C \in CO\text{-}NPC$

3.5.2. Ejercicio 8

Sea el lenguaje $SH\text{-}s\text{-}t = \{(G, s, t) | G$ es un grafo que tiene un camino (o sendero) de Hamilton del vértice s al vértice $\}$. Un grafo $G = (V, E)$ tiene un camino de Hamilton del vértice s al vértice t sii G tiene un camino entre s y t que recorre todos los vértices restantes una sola vez. Probar que SH-s-t es NP-completo. Ayuda: Probar con una reducción desde el lenguaje CH, el lenguaje correspondiente al problema del circuito hamiltoniano, que es NP-completo.

Se probará que SH-s-t es NPC, encontrando una reducción de CH a SH-s-t.

1. Función de reducción

Se define f(G) = (G', s, t), donde G' es el mismo grafo con la diferencia de que se agrega un vértice t que duplica los arcos de algún vértice s del grafo G.

2. Computabilidad polinomial

Existe una MT M_f que toma el vértice s del grafo G con menos arcos y agrega un vértice t con los mismos arcos del vértice s, lo cual es simplemente agregar un nuevo nodo copiando los arcos de v, de esta manera generando G', y codificando posteriormente s y t a la entrada generando la salida (G', s, t).

Entonces como el proceso es encontrar el vértice s con menos arcos, copiar los arcos del vértice s para el nuevo vértice t y luego agregarlo, esto podría acotarse como: $O(|E| * |V|) = O(|G|) = O(n) = poly(n)$

3. Correctitud

- $G \in CH \Rightarrow G' \in SH\text{-}s\text{-}t$, esto es verdadero, debido a que si G tiene un circuito de hamilton, desde cada vértice en el ciclo, hay un camino que vuelve al mismo vértice, ya que el ciclo permite regresar al punto de inicio. Lo que implica que si duplicamos los arcos de un vértice s para un nuevo vértice t, habrá un camino de hamilton de s a t, en consecuencia (G', s, t) pertenece a SH-s-t
- $G \notin CH \Rightarrow G' \notin SH\text{-}s\text{-}t$, esto es verdadero, debido a que si G no tiene un circuito de hamilton, por más que duplique los nodos no estaría formando un ciclo ya el vértice t solo agregaría los mismos caminos que s, que no permitieron la formación de un ciclo. También para el caso de que no hubiera un ciclo pero si un camino, al tomar el vertice s con menos arcos para generar t, estoy limitando que se elija un vértice que podría generar un ciclo, ya que si se toma un vértice que tuviera un camino a cada vértice al duplicarlo estaría generando un ciclo que no existía.

3.5.3. Ejercicio 9

Sea el lenguaje $SUB\text{-}ISO = \{(G1, G2) | G1$ es isomorfo a un subgrafo de $G2\}$. Se define que dos grafos son isomorfos si son idénticos salvo por el nombre de sus arcos. Probar que SUB-ISO es NP-completo. Ayuda: Probar con una reducción polinomial desde el lenguaje CLIQUE, el lenguaje correspondiente al problema del clique, que es NP-completo.

Se probará que SUB-ISO pertenece a NPC, encontrando una reducción de CLIQUE a SUB-ISO

1. Función de reducción

Se define $f(G, K) = (G', G)$, donde G' es el subgrafo completo de tamaño K.

2. Computabilidad polinomial

Existe una MT M_f que genera un grafo completo de tamaño K y lo concatena al grafo G para generar la salida (G', G). Entonces como el proceso es generar un grafo completo de tamaño K, a lo sumo deben generarse K vertices y K-1 arcos, siendo que K <|V|, sino la máquina computaría una salida inválida, entonces puede acotarse a $O(K * K - 1) = O(|V| * |V|) = O(|G|^2)$

3. Correctitud

Sea G_k el subgrafo completo de tamaño K

- $(G, K) \in CLIQUE \Rightarrow G_K$ es un subgrafo de G $\Rightarrow G'$ es isomorfo a un subgrafo de G $\Rightarrow (G', G) \in SUB\text{-}ISO$
- $(G, K) \notin CLIQUE \Rightarrow G_K$ no es un subgrafo de G $\Rightarrow G'$ no es isomorfo a un subgrafo de G $\Rightarrow (G', G) \notin SUB\text{-}ISO$

3.6. Complejidad Espacial

3.6.1. Ejercicio 10

Probar que el problema de los palíndromos está en $SPACE(n)$.

El problema de palíndromos consiste en determinar si una palabra se lee igual de izquierda a derecha o viceversa. Entonces es posible definir una MT que realice el siguiente proceso:

1- Se toma la primera letra y luego se recorre el input hasta llegar al final de la palabra.

- Si la última letra coincide con la primera significa que parcialmente es palíndromo. Entonces se borran ambas letras y se vuelve a repetir el paso (1).

- Si no hay última letra significa que la letra inicial era la última implicando que la palabra era palíndromo ya que se analizaron todos los pares de letras y estas eran iguales o porque la palabra tenía una cantidad de letras impares. Entonces se acepta el input.
- Si las letras no coinciden la palabra no es palíndromo. Entonces se rechaza el input.

Como este algoritmo, como puede observarse, se usa una cantidad de espacio lineal (más allá del input no se utilizan celdas adicionales), no excede O(n) espacio de memoria (n siendo tamaño de entrada), por lo tanto el problema está en SPACE(n)

3.6.2. Ejercicio 11

Justificar por qué el lenguaje $QSAT$ no pertenecería a P ni a NP.

QSAT es el problema de satisfacción de fórmulas cuantificadas, donde se intenta definir si una fórmula booleana cuantificada es satisfactible. Para saber si no pertenecería a P ni a NP hay que tener en cuenta la complejidad cuantificada de este problema, la cual introduce una gran complejidad puesto que se necesita evaluar todas las combinaciones posibles de asignaciones de variables para los cuantificadores. Esto ya nos muestra la naturaleza exponencial del problema, ya que habría que ver todas las posibles combinaciones de valores de verdad para estas variables cuantificadas.

- Si hay un cuantificador universal ($\forall x$), debemos verificar que la fórmula es verdadera para todas las posibles asignaciones de x.
- Si hay un cuantificador existencial ($\exists x$), debemos verificar que la fórmula es verdadera para al menos una asignación de x.
- Dado que cada variable puede tomar los valores Verdadero (V) o Falso (F), si tenemos n variables, el número de evaluaciones posibles en el peor caso es: 2^n

Es decir, por culpa de la naturaleza exponencial de este problema, no puede resolverse con algún algoritmo en tiempo polinomial, entonces QSAT no pertenecería a P, pero tampoco a NP ya que para el caso existencial solo sería necesario analizar una asignación satisfactible, en cambio para el cuantificador universal sería necesario analizar todas las asignaciones y que efectivamente todas estas (2^n posibilidades) sean satisfactibles.

3.6.3. Ejercicio 12

Probar que $NP \subseteq PSPACE$. Ayuda: Si L pertenece a NP, existe una MT M1 capaz de verificar en tiempo poly(|w|) si una cadena w pertenece a L, con la ayuda de un certificado x de tamaño poly(|w|). De esta manera, se puede construir una MT M2 que decide L en espacio poly(|w|).

Idea general

Para demostrar que $NP \subseteq PSPACE$, es necesaria la construcción de una MT M que decida un lenguaje $L \in NP$ en espacio PSPACE. Sabiendo que $L \in NP$, existe una MTN M' que genera aleatoriamente un certificado y la valida, donde todas sus computaciones tardan tiempo poly(n).

Así, se puede generar una MT M que emula cada rama de M', que reutiliza el mismo espacio cada vez que genera una rama, es decir que una vez terminada una rama, borra la cinta para generar la siguiente.

Construcción de la MT

Sea w la cadena de entrada, M funcionaría de la siguiente manera:

1. Se emula una rama de M' con entrada w
2. De acuerdo a la emulación de la rama:
    - Si la emulación se detiene en q_A, M para en q_A

- Si la emulación se detiene en q_R, M limpia la cinta y vuelve al paso (1) para emular la siguiente rama.
- Si se exploraron todas las ramas sin llegar a q_A, M para en q_R

Correctitud

- $w \in L \Rightarrow (\exists r)(r : r$ es una rama que acepta w$) \Rightarrow w \in L(M)$
- $w \notin L \Rightarrow \neg(\exists r)(r : r$ es una rama que acepta w$) \Rightarrow w \notin L(M)$

Análisis espacial

Como cada rama de M' se resuelve en tiempo poly(n), a lo sumo pueden ocuparse poly(n) celdas, y como M borra la cinta antes de empezar una nueva rama, a lo sumo podrá ocupar poly(n) celdas, por lo cual M trabajo en espacio poly(n).

$\therefore NP \subseteq PSPACE$