Pauta de Corrección

Segundo Certamen
Informática Teórica

22 de junio de 2024

1. No es decidible. Reducimos el problema ACCEPTS (¿La máquina de Turing M acepta σ?) a este problema. Sabemos que ACCEPTS no es decidible. Construimos una máquina de Turing M' que acepta Σ^*, y preguntamos si hay σ tal que $\sigma \in \mathscr{L}(M)$ y que $\sigma \in \mathscr{L}(M')$. Decidir esto es decidir ACCEPTS, que sabemos imposible.

   Alternativamente, tomando $M_1 = M_2$, decidir si hay ω tal que $\omega \in \mathscr{L}(M_1) \wedge \omega \in \mathscr{L}(M_2)$ es lo mismo que decidir si $\mathscr{L}(M_1) = \varnothing$, que sabemos imposible.

Puntajes

Total	25
Plantear una reducción desde un problema no decidible	5
Demostrar la reducción	20

2. Una solución es escribir programas que hagan las tareas correctamente, justificando que siempre se ejecutan en tiempo finito. Usaremos C++ al efecto. Dados lenguajes L_1 y L_2, suponemos funciones bool f1(std :: string sigma); y bool f2(std :: string sigma); que deciden si sigma pertenece a L_1 o L_2, respectivamente. Entonces:

   a) Para reconocer si sigma está en $L_1 \cup L_2$ podemos escribir la función del listado 1.

```
bool in_union(std::string sigma)
{
    return f1(sigma) || f2(sigma);
}
```

   Listado 1: Decidir unión $L_1 \cup L_2$

   Es claro que si f1(sigma) y f2(sigma) siempre terminan dando la respuesta correcta, la función in_union(sigma) siempre termina dando la respuesta correcta.

   b) Para reconocer si sigma está en $L_1 \cdot L_2$ podemos escribir la función del listado 2.

```
bool in_concatenation(std::string sigma)
{
    for(int k = 0; k <= sigma.length(); ++k) {
        std::string sigma1 = sigma.substr(0, k);
        std::string sigma2 = sigma.substr(k);
        if(f1(sigma1) && f2(sigma2))
            return true;
    }
    return false;
}
```

   Listado 2: Decidir concatenación $L_1 \cdot L_2$

   Es claro que si f1(sigma) y f2(sigma) siempre terminan dando la respuesta correcta, la función in_concatenation(sigma) siempre termina dando la respuesta correcta, prueba un número finito de subdivisiones de sigma.

Puntajes

Total			30
a) Unión		15	
Construcción clara	8		
Justificar tiempo finito	7		
b) Concatenación		15	
Construcción clara	8		
Justificar tiempo finito	7		

3. Cada uno por turno, con una breve explicación. Consideramos problemas D_i decidibles, I_i no decidibles, E_i computacionalmente enumerables no decidibles.

   a) $\boldsymbol{I_1 \leq D_1}$**:** Esto es imposible. Si fuera posible, podríamos decidir si $\sigma \in I_1$ aplicando la reducción y decidiendo si el resultado pertenece a D_1.

   b) $\boldsymbol{E_2 \leq I_2}$**:** Posible. Los lenguajes computacionalmente enumerables no decidibles incluso son un subconjunto de los no decidibles.

   c) $\boldsymbol{\overline{E_3} \leq E_4}$**:** Si E_3 es enumerable no decidible, su complemento no es ni siquiera enumerable. Esto es imposible.

   d) $\boldsymbol{\overline{D_2} \leq D_3}$**:** Si D_2 es decidible, lo es su complemento $\overline{D_2}$. Esto es posible.

Puntajes

Total	20
a) Explicación y conclusión	5
b) Explicación y conclusión	5
c) Explicación y conclusión	5
d) Explicación y conclusión	5

4. Cada uno por turno, con una breve explicación. Consideramos problemas $P_i \in \mathsf{P}$, $N_i \in \mathsf{NP}$, C_i es NP-completo, y X_i es desconocido; queremos saber qué se puede concluir sobre X_i en cada caso.

   a) **$X_1 \leq_p N_1$ y $P_1 \leq_p X_1$:** De $X_1 \leq_p N_1$ tenemos que X_1 está en NP, $P_1 \leq_p X_1$ no aporta información adicional.

   b) **$C_2 \leq X_2$:** Nada podemos concluir, ya que la reducción no es polinomial.

   c) **$X_2 \leq P_2$ y $X_2 \leq_p C_2$:** De $X_2 \leq P_2$ solo podemos concluir que X_2 es decidible (los problemas en P son decidibles), $X_2 \leq_p C_2$ permite concluir que X_2 está en NP.

   d) **$C_4 \leq_p X_4$ y $X_4 \leq_p N_4$:** De $C_4 \leq_p X_4$ concluimos que X_4 es NP-duro, de $X_4 \leq_p N_4$ sabemos que X_4 está en NP; con lo que X_4 es NP-completo.

Puntajes

Total	20
a) Explicación y conclusión	5
b) Explicación y conclusión	5
c) Explicación y conclusión	5
d) Explicación y conclusión	5

5. Para demostrar que DOUBLE SAT es NP-completo debemos demostrar que está en NP y que es NP-duro.

   Para demostrar que está en NP, un certificado es dos juegos de valores de las variables de ϕ, llamémosles $\mathbf{x}$ y $\mathbf{x}'$. Debemos verificar que $\mathbf{x} \neq \mathbf{x}'$, lo que es posible de hacer en tiempo $O(n)$; y verificar que $\phi(\mathbf{x})$ y $\phi(\mathbf{x}')$ son ambos verdaderos, cosa que toma tiempo proporcional al largo de ϕ en un lenguaje de programación como Python.

   Para demostrar que es NP-duro, reducimos desde el problema SAT: Dada una fórmula $\phi(x_1, \ldots, x_n)$, agregamos una nueva variable z y creamos la fórmula:

   $$\phi'(x_1, \ldots, x_n, z) = (\phi(x_1, \ldots, x_n)) \wedge (z \vee \overline{z})$$

   Es claro que si se puede satisfacer $\phi(x_1, \ldots, x_n)$, entonces $\phi'(x_1, \ldots, x_n, z)$ puede satisfacerse de al menos dos formas (con z verdadero o falso); si $\phi(x_1, \ldots, x_n)$ no puede satisfacerse, tampoco puede satisfacerse $\phi'(x_1, \ldots, x_n, z)$, en particular, no se satisface de dos formas. Es una reducción, y la traducción claramente se efectúa en tiempo polinomial.

   Como está en NP y es NP-duro, es NP-completo.

Puntajes

Total		30
Hay que demostrar NP y NP-duro		4
Demostrar que está en NP		6
Demostrar que es NP-duro		20
— Detalles de la reducción	15	
— Justificar que es polinomial	5	