Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2213 - Lógica para Ciencias de la Computación
Profesor: Miguel Romero
Ayudantes: Caetano Borges, Valentina Juri

Ayudantía 9

Repaso I2

05 de junio de 2026

Problema 1. Considere dos teorías Σ_1 y Σ_2 sobre un mismo vocabulario $\mathcal{L}$. Demuestre que $\Sigma_1 \cap \Sigma_2$ es también una teoría. ¿Es siempre cierto que $\Sigma_1 \cup \Sigma_2$ es una teoría?

Solución: Primero demostraremos que $\Sigma_1 \cap \Sigma_2$ es consistente. Supongamos por contradicción que no lo es. Entonces $\Sigma_1 \cap \Sigma_2 \vdash \alpha$, con α una contradicción (oración no satisfacible). Como $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_1$, esto implica que $\Sigma_1 \vdash \alpha$. Pero entonces Σ_1 sería inconsistente, lo que contradice el hecho de que sea una teoría. Esto demuestra que $\Sigma_1 \cap \Sigma_2$ es consistente.

Ahora veamos que $\Sigma_1 \cap \Sigma_2$ es cerrado bajo consecuencia lógica. Para ello, sea φ una $\mathcal{L}$-oración tal que $\Sigma_1 \cap \Sigma_2 \models \varphi$. Como $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_1$ y $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_2$, se cumple que $\Sigma_1 \models \varphi$ y $\Sigma_2 \models \varphi$. Lo anterior se debe a que las $\mathcal{L}$-estructuras que satisfacen a Σ_1 y Σ_2, satisfacen a cualquiera de sus subconjuntos, así, como $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_1$ y $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_2$, estas satisfacen a $\Sigma_1 \cap \Sigma_2$. Como Σ_1 es cerrado bajo consecuencia lógica (es una teoría), tenemos que $\varphi \in \Sigma_1$, análogamente tenemos que $\varphi \in \Sigma_2$. Luego $\varphi \in \Sigma_1 \cap \Sigma_2$. Esto prueba que $\Sigma_1 \cap \Sigma_2$ es una teoría.

Véase ahora un ejemplo en el que $\Sigma_1 \cup \Sigma_2$ no es una teoría. Se $\mathcal{L}$ un vocabulario con un símbolo de constante c y un predicado unario P. Notemos que $\{P(c)\}$ y$\{\neg P(c)\}$ son consistentes, por lo que podemos considerar las teorías $\Sigma_1 = \text{Th}(\{P(c)\})$ y $\Sigma_2 = \text{Th}(\{\neg P(c)\})$. Sin embargo $\Sigma_1 \cup \Sigma_2$ no puede ser una teoría, puesto que es inconsistente.

Problema 2. Sea Σ una teoría. Demuestre que Σ es completa sí y solo sí para todo par de estructuras $\mathfrak{A}$ y $\mathfrak{B}$ se cumple que:

$$\mathfrak{A} \models \Sigma \text{ y } \mathfrak{B} \models \Sigma \rightarrow \mathfrak{A} \text{ y } \mathfrak{B} \text{ son elementalmente equivalentes}$$

Solución: ($\Rightarrow$) Sean $\mathfrak{A}$ y $\mathfrak{B}$ estructuras arbitrarias tales que $\mathfrak{A} \models \Sigma$ y $\mathfrak{B} \models \Sigma$ con Σ una teoría completa. Debemos demostrar que $\mathfrak{A}$ y $\mathfrak{B}$ son elementalmente equivalentes. Es decir, sea φ una o¡ración arbitraria, debemos mostrar que:

$$\mathfrak{A} \models \varphi \iff \mathfrak{B} \models \varphi$$

Equivalente a:

$$\mathfrak{A} \not\models \varphi \iff \mathfrak{B} \not\models \varphi$$

Supongamos que $\mathfrak{A} \nvDash \varphi$ (la otra dirección es simétrica). Como $\mathfrak{A} \models \Sigma$, tenemos que $\varphi \notin \Sigma$. Por completitud de Σ, sigue que $\neg\varphi \in \Sigma$. Como $\mathfrak{B} \models \Sigma$, obtenemos que $\mathfrak{B} \models \neg\varphi$, es decir, $\mathfrak{B} \nvDash \varphi$.

($\Leftarrow$) Debemos demostrar que Σ es completa. Por contradicción, supongamos que existe una oración φ tal que $\varphi \notin \Sigma$ y $\neg\varphi \notin \Sigma$. Como toda teoría es cerrada bajo consecuencia lógica, esto es equivalente a $\Sigma \nvDash \varphi$ y $\Sigma \nvDash \neg\varphi$. Por definición de consecuencia lógica existe una estructura $\mathfrak{A}$ tal que $\mathfrak{A} \models \Sigma$ y $\mathfrak{A} \nvDash \varphi$, y existe una estructura $\mathfrak{B}$ tal que $\mathfrak{B} \models \Sigma$ y $\mathfrak{B} \nvDash \neg\varphi$. Así, obtenemos dos estructuras $\mathfrak{A}$ y $\mathfrak{B}$ tal que $\mathfrak{A} \models \Sigma$ y $\mathfrak{B} \models \Sigma$, pero que no son elementalmente equivalentes, ya que $\mathfrak{A} \nvDash \varphi$ $\mathfrak{B} \models \varphi$. Esto contradice nuestra hipótesis.

Dado lo anterior, queda demostrada la propiedad pedida.

Problema 3. Demuestre que el siguiente lenguaje de pares enteros (entregados en binario) está en P:

$$\textsc{RelPrimes} := \{\langle a, b\rangle \mid a, b \geq 1 \text{ y } a \text{ y } b \text{ no tienen un divisor común mayor a } 1\}$$

Solución: Podemos usar el algoritmo de Euclídes para calcular el máximo común divisor del par $\langle a, b\rangle$, el cual se presenta a continuación.

---

1. Inicializar x:= máx(a,b)
2. Inicializar y:= mín(a,b)
3. While $(y > 0)$:
    4. Reasignar x = x mod y
    5. Intercambiar valores de x e y.
6. Retornar x.

Omitiremos la demostración de que el algoritmo de Euclídes es correcto. Lo que debemos hacer es ejecutar el algoritmo y verificar si el número que retorna es o no igual a 1. Cada línea del pesudocódigo puede hacerse en tiempo polinomial (queda de ejercicio demostrar que el algoritmo aprendido en el colegio para dividir dos números acotados superiormente por n funciona en $\mathcal{O}(\log(n)^2)$). Lo que queda por probar es que el bucle entre las líneas 3 y 5 se ejecuta un número polinomial de veces en términos de $\log(a) + \log(b)$, correspondiente al largo de la entrada. Para probar esto, mostraremos que cada ejecución de la línea 4 reduce el valor de x al menos en la mitad, de lo que seguirá que el número de iteraciones posibles no puede ser mayor a $\log(a) + \log(b)$. Notemos que, por definición del resto de la división, al principio y al final de cada iteración tenemos que $x \geq y$. Tenemos dos casos:

- Si $\frac{x}{2} \geq y$, entonces $x \bmod y < y \leq \frac{x}{2}$. Esto dado que el resto de una división es menor que el divisor ($x \bmod y < y$). Por lo tanto, el valor de x disminuyó a menos de la mitad.
- Si $\frac{x}{2} < y$, entonces $x \bmod y = x - y < \frac{x}{2}$. Esto dado que $x - 2y < 0$ y $x - 0y \geq y$, de modo que $x - y$ es el resto. Por lo tanto, el valor de x disminuyó a menos de la mitad.

Dado lo anterior, se concluye que el algoritmo es polinomial en $\log(a) + \log(b)$, por lo que $\textsc{RelPrimes} \in P$.

Problema 4. Dada una paleta de colores $S = \{c_1, c_2, ..., c_n\}$ y un grafo no dirigido $G = (V, E)$, decimos que una función $f : V \to S$ es una coloración de G con colores del conjunto S, si para todo par de

nodos $u, v \in V$, tales que $(u, v) \in E$, $f(u) \neq f(v)$. Es decir, f corresponde a una asignación de colores de S a los nodos de G, que asigna colores distintos a nodos adyacentes entre sí.

Un grafo no dirigido G es k-coloreable (con $k \in \mathbb{N}$), si existe una coloración para G con colores de S, para $|S| = k$. En base a lo anterior, demuestre que el siguiente lenguaje está en P:

$$\text{2-COL} := \{\langle G \rangle \mid \text{G grafo no dirigido 2-coloreable}\}$$

Solución: Se propone el siguiente algoritmo que resuelve 2-COL en tiempo polinomial. Sea $G = (V, E)$, con $V = \{v_1, ..., v_n\}$. Véase que si existe un 2-coloreo, este algoritmo lo retorna en el arreglo **colores**.

---

1. Inicializar arreglos **colores** y **visitados** de tamaño n, con todos sus valores en -1.
2. Inicializar arreglo **por-revisar** vacío.
3. Asignar primer color (c_1) a primer nodo (v_1). Marcar nodo como visitado (**visitados**$[v_1] = 1$).
4. Insertar (al final del arreglo) vecinos de v_1 a arreglo **por-revisar**. Marcar vecinos como visitados.
5. Mientras la lista **por-revisar** no esté vacía:
    6. Extraer primer elemento: v.
    7. Revisar los colores de los vecinos de v. Si todos son de un mismo color, asignar a v el color contrario. Si hay dos vecinos de colores distintos, **retornar False**.
    8. Insertar vecinos no visitados a arreglos **por-revisar**. Marcar vecinos como visitados.
9. **Retornar True**.

Nótese que este algoritmo se basa en BFS y que funciona bajo la hipótesis de que todos los nodos están conectados entre sí, es decir, que siempre existe un camino desde un nodo a otro. La versión generalizada implica repetir desde el paso 3 en adelante para todos los nodos no revisados, garantizando que se recorra todo el grafo. Esta versión hace que la complejidad sea n veces mayor (se debe repetir el algoritmo para cada nodo en el peor caso), por lo que sigue siendo polinomial.

Consideremos así, que existe un camino entre cualquier par de nodos. Se deja como ejercicio demostrar la correctitud de este algoritmo. Respecto a la complejidad, nótese que el paso (1) es lineal en la cantidad de nodos (dado que esta entrega el tamaño de los arreglos) y los pasos (2) y (3) es constante. El paso (4) es $\mathcal{O}(n)$, dado que en el peor caso, el nodo inicial podría estar conectado a todos los demás. El paso (6) es constante o lineal dependiendo de la implementación (puede implicar un movimiento de una posición hacia adelante de todos sus elementos). El paso (7) es polinomial en n por la revisión de los vecinos. Así, verificar el color de cada vecino es $\mathcal{O}(1)$ y para corroborar que todos los vecinos sean del mismo color solo se necesita una variable auxiliar que almacene el color visto (Nótese que esto aplica solo para los nodos que poseen un color, si alguno no tienen un color asignado, se ignora). Nuevamente el paso (8) es lineal en n.

Finalmente nos falta argumentar que el ciclo en (5) se ejecuta un número polinomial de veces. Nótese que cada nodo se agrega a la lista por-revisar a lo más una vez (por pasos 4. y 8.) con estado de revisado, el arreglo por-revisar tiene a lo más n elementos, dada la cantidad de nodos. Así, como el ciclo continúa mientras la lista por-revisar no esté vacía y en cada iteración se elimina un elemento de la lista en el paso (6), el ciclo se ejecuta $\mathcal{O}(n)$ veces. Por lo tanto, concluimos así que el algoritmo completo es polinomial en n. Por lo que 2-COL $\in P$.