Pregunta 1

1.

Se tiene un algoritmo A de caja negra de resolución SAT, es decir, un dispositivo que toma una fórmula de lógica proposicional ϕ, A(ϕ) es verdadero si ϕ es satisfacible.

a) Algoritmo que utiliza A como subrutina para determinar si ϕ es una tautología:

```
def esTautologia(p):
    if A(~p) == 0: #Si ~p es insatisfacible
        return True
    else:
        return False
```

Probaremos si el algoritmo anterior es correcto:

Demostración. PDQ: $\neg\phi$ es insatisfacible $\Rightarrow \phi$ es tautología

$$\begin{aligned} \neg\phi \text{ es insatisfacible} &\Rightarrow (\emptyset \cup \{\neg\phi\}) \text{ es insatisfacible} \\ &\Rightarrow \emptyset \models \phi \\ &\Rightarrow \sigma(\phi) = 1 \\ &\Rightarrow \phi \text{ es Tautología} \end{aligned}$$

□

b) Se tienen 2 fórmulas proposicionales ϕ y φ, se va a determinar si tienen los mismos valores de verdad. Algoritmo que utiliza A para responder a esta pregunta:

```
def equivalentes(p, q):
    if A(~((~p or q) and (p or ~q))) == 0:
        return True
    else:
        return False
```

Probaremos si el algoritmo anterior es correcto:

Demostración. PDQ: $\neg((\neg\phi \vee \varphi) \wedge (\phi \vee \neg\varphi))$ es insatisfacible $\Rightarrow \phi \equiv \varphi$

$$\begin{aligned} \neg((\neg\phi \vee \varphi) \wedge (\phi \vee \neg\varphi)) \text{ es insatisfacible} &\Rightarrow (\neg\phi \vee \varphi) \wedge (\phi \vee \neg\varphi) \text{ es Tautología} \\ &\Rightarrow (\phi \Rightarrow \varphi) \wedge (\varphi \Rightarrow \phi) \text{ es Tautología} \\ &\Rightarrow (\phi \Leftrightarrow \varphi) \text{ es Tautología} \\ &\Rightarrow \sigma(\phi) = \sigma(\varphi) \\ &\Rightarrow \phi \equiv \varphi \end{aligned}$$

□

c) Se tiene una fórmula proposicional ϕ con n variables que se sabe que es satisfacible. Algoritmo que utiliza A como subrutina para obtener una asignación satisfactoria para ϕ utilizando como máximo n llamadas a A:

Sean $\phi_1, ..., \phi_n$ proposiciones de la fórmula, luego sea ϕ' tal que $\sigma(\phi_1) = 1$

Si $A(\phi')$ satisfacible $\Rightarrow \sigma(\phi_1) = 1$
Si $A(\phi')$ insatisfacible $\Rightarrow \sigma(\phi_1) = 0$

Luego ϕ'' tal que $\sigma(\phi_1) = \{\text{valor con el que es satisfacible}\}$ y $\sigma(\phi_2) = 1$, se repite el proceso anterior n veces hasta encontrar todas las valuaciones $\sigma(\phi_k)$, $k \in 1, ..., n$.

Demostración. Para probar la correctitud se procedera por el principio de inducción:

$\boxed{\text{Caso Base:}}$

Sea ϕ una formula lógica con ϕ_1 su única proposición y se sabe que existe $\sigma(\phi_1)$ tal que $\sigma(\phi) = 1$. Sea entonces ϕ' tal que $\sigma(\phi_1) = 1$, como la fórmula depende solo de ϕ_1, si con $\sigma(\phi) = 1$ se cumple que $A(\phi')$ entrega satisfacible también se cumple que $\sigma(\phi') = 1$, por el contrario si $A(\phi')$ es insatisfacible entonces vasta con tomar $\sigma(\phi) = 0$, con esa vaulacion se tendria el otro caso.

$\boxed{\text{Caso Inductivo:}}$

Supongamos que existen n-1 valuaciones $\sigma(\phi_1), ..., \sigma(\phi_{n-1})$ con las que se cumple $A(\phi^{(n-1)})$ entrega satisfacible. Como se sabe que $\phi^{(n-1)}$ es satisfacible entonces debe existir una valuacion $\sigma(\phi_n)$ tal que $\sigma(\phi^{(n-1)}) = 1$. Entonces para encontrar la n-ésima valuación vasta con tomar $\phi^{(n)}$ tal que $\sigma(\phi_n) = 1$, si con esa valuacion $A(\phi^{(n)})$ entrega que es satisfacible entonces encontramos todas las valuaciones con las que $\sigma(\phi) = 1$. En caso de que $A(\phi_{(n)})$ entregue insatisfacible, vasta tomar $\sigma(\phi_n) = 0$.

$\square$

El algoritmo anterior nos entrega las n valuaciones de las variables de la formula proposicional con solo n llamadas.

2.

Para que $\sigma(\phi_i) = 1$ se necesita que $\sigma(a_i) = 1$ o $\sigma(b_i) = 1$, pero si ambos valoran 1 entonces $\sigma(\phi_i) = 0$ y $\sigma(\phi_{i+1}) = 1$. Definimos $p \wedge q = \neg(\neg p \vee \neg q)$.

Se tiene que existen 3 casos posibles para que $\sigma(\phi_i) = 1$:

- Que ($\sigma(a_{i-1}) = 0$, $\sigma(b_{i-1}) = 0$ ó $\sigma(a_{i-1}) = 1$, $\sigma(b_{i-1}) = 0$ ó $\sigma(a_{i-1}) = 0$, $\sigma(b_{i-1}) = 1$) y ($\sigma(a_i) = 1$, $\sigma(b_i) = 0$) ó ($\sigma(a_i) = 0$, $\sigma(b_i) = 1$)

  Donde lógicamente esto se escribe:

  $[((a_i \wedge \neg b_i) \vee (\neg a_i \wedge b_i)) \wedge \neg(a_{i-1} \wedge b_{i-1})]$

  $= [(a_i \wedge \neg b_i) \wedge \neg(a_{i-1} \wedge b_{i-1})] \vee [(\neg a_i \wedge b_i) \wedge \neg(a_{i-1} \wedge b_{i-1})]$

  $= [a_i \wedge \neg(b_i \vee (a_{i-1} \wedge b_{i-1}))] \vee [b_i \wedge \neg(a_i \vee (a_{i-1} \wedge b_{i-1}))]$

- Que $\sigma(a_{i-1}) = 1$, $\sigma(b_{i-1}) = 1$ y $\sigma(a_i) = 0$, $\sigma(b_i) = 0$

  Donde lógicamente esto se escribe:

  $(\neg a_i \wedge \neg b_i) \wedge (a_{i-1} \wedge b_{i-1}) = [(a_{i-1} \wedge b_{i-1}) \wedge \neg(a_i \vee b_i)]$

- Que $\sigma(a_{i-1}) = 1$, $\sigma(b_{i-1}) = 1$ y $\sigma(a_i) = 1$, $\sigma(b_i) = 1$

  Donde lógicamente esto se escribe:

  $[a_i \wedge b_i \wedge (a_{i-1} \wedge b_{i-1})]$

Luego la unión de todos estos casos y además definiendo los casos de los extremo, queda dado por:

Caso Base: $i = 0$

$$\sigma(\phi_0) = \sigma((a_0 \wedge \neg b_0) \vee (b_0 \wedge \neg a_0)) \tag{1}$$

Caso Intermedio: $i = 1, ..., n-1$

$$\sigma(\phi_i) = \sigma \begin{pmatrix} [a_i \wedge \neg(b_i \vee (a_{i-1} \wedge b_{i-1}))] \\ \vee[b_i \wedge \neg(a_i \vee (a_{i-1} \wedge b_{i-1}))] \\ \vee[(a_{i-1} \wedge b_{i-1}) \wedge \neg(a_i \vee b_i)] \\ \vee[a_i \wedge b_i \wedge (a_{i-1} \wedge b_{i-1})] \end{pmatrix} \tag{2}$$

Caso Final: $i = n$

$$\sigma(\phi_n) = \sigma(a_{n-1} \wedge b_{n-1}) \tag{3}$$