Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
Primer Semestre de 2022

IIC 2213 – Lógica para ciencia de la Computación
Tarea 4 - Entrega Viernes 06 de Mayo a las 20:00 - via canvas

Recuerda que esta tarea es individual. Puedes discutir sobre la respuesta con tus compañeros (¡y eso está muy bien!), pero no puedes enviar la respuesta a nadie o utilizar la respuesta de alguien más. ¡Aprovecha esta tarea para combinar lo que sabes de algoritmos y programación con lo que has aprendido de lógica!

Sistemas de conocimiento negativo Sea P un conjunto de proposiciones. Un sistema de conocimiento negativo sobre P es un conjunto Γ de fórmulas en $L(P)$, que pueden ser de una de estas dos formas:

- Implicancias de la forma $p \rightarrow q_1 \vee \cdots \vee q_n$, para $p, q_1, \ldots, q_n$ en P
- Fórmulas $q_1 \vee \cdots \vee q_n$, para $q_1, \ldots, q_n$ en P.
- Fórmulas $\neg p$, para $p \in P$

Tarea

1. (no se evalúa) Diseña un algoritmo para determinar cuando un sistema de conocimiento negativo Γ es satisfacible, pero que además funcione tiempo polinomial. **Idea**: Transforma cada fórmula en Γ a CNF. ¿Cuándo podría no ser satisfacible este conjunto de cláusulas? ¿Cuándo puedes determinar que una propocisión debe necesariamente ser valuada como verdadero o como falso para satisfacer ese conjunto de cláusulas?

2. Programa tu algoritmo en python, asumiendo que tu sistema de conocimiento viene dado como en CNF, según la simplificación del formato DIMACS que describimos abajo.

3. Argumenta por que el algoritmo que programaste funciona en tiempo polinomial con respecto al tamaño de Γ, o más específicamente, con respecto al tamaño de la codificación de Γ en formato DIMACS (el número de símbolos).

Formato DIMACS Tu programa solo va a recibir fórmulas en CNF. El formato de las formulas es una versión simplificada de DIMACS, el formato que se usa normalmente en solvers profesionales. Acá va un archivo de ejemplo, que codifica al sistema $(x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee x_4)$:

```
c esto es un comentario
c
c esto también es un comentario
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 4 0
%
0
```

Las reglas del input son:

- Al principio hay un numero arbitrario de líneas que comienzan con c, todas estas son comentarios y deben ignorarse.
- La primera línea siempre es `p cnf <nprop> <nclaus>`, donde `<nprop>` es el número de proposiciones y `<nclaus>` es el número de cláusulas de la fórmula. Puedes asumir que todas las variables siempre aparecen en al menos una cláusula.
- Luego es una línea por cláusula. Cada una de estas es una secuencia de números entre $[-\texttt{nprop}, -1] \cup [1, \texttt{nprop}]$, y que termina en un 0. Un número positivo representa a la proposición asociada a ese número, y un número negativo a la negación de esa proposición.
- Puedes asumir también que los números no se repiten en una misma cláusula, ni tampoco las cláusulas contienen un número y su negativo al mismo tiempo.
- Finalmente, el archivo termina con un caracter % seguido de un 0.

Código

Importante. No puedes usar librerías con funciones específicas para el manejo de lógica o de solvers para lógica.

Debes generar un código en python donde se vean claramente dos funciones:

- Una función dimacs(texto) que reciba el nombre de un archivo que codifica una formula (se asume que el archivo está en la misma carpeta que el código), lo abra y lo transforme en un formato a tu elección (arreglo, lista, diccionario, lo que sea), que va a contener tu fórmula ya procesada.
- Una función decidirSAT(formula) que reciba una fórmula ya procesada y entregue un 1 si la fórmula es satisfacible o un 0 si no lo es.

Formato de entrega La entrega debe ser una carpeta (comprimida) en donde se encuentre un archivo tarea_nombre_apellido.py, junto a un archivo de texto o pdf donde se argumente por qué el algoritmo funciona en tiempo polinomial.