PROGRAMACIÓN DECLARATIVA CURSO 2022-23
PRÁCTICA INDIVIDUAL HASKELL

Objetivo de la práctica: Escribir un programa en Haskell para procesar fórmulas proposicionales, reconocer Cláusulas de Horn y trabajar con el método deductivo de Resolución.

Descripción del problema: Consideramos la signatura $\Sigma = \{P, Q, R, S\}$. Una Σ-fórmula de la lógica proposicional con esta signatura puede ser atómica, si es un símbolo de proposición de Σ, o compuesta, si está formada con las reglas de formación de fórmulas usando los operadores lógicos $\neg$, $\wedge$, $\vee$. Por ejemplo, las siguientes son Σ-fórmulas de la lógica proposicional:

$f_1 \equiv (P \wedge \neg Q) \vee \neg S$
$f_2 \equiv (\neg R \vee P) \wedge \neg(Q \vee \neg P)$
$f_3 \equiv (S \wedge Q) \wedge (\neg P \vee R) \wedge S$

Recuerda que un *literal*, L, es un átomo o la negación de un átomo, que una *cláusula (disyuntiva)*, C, es una disyunción de literales y que una fórmula está en *forma normal conjuntiva* (FNC) si es conjunción de cláusulas.

Dadas dos cláusulas $C_1 = L_1^1 \vee \ldots \vee L_n^1$ y $C_2 = L_1^2 \vee \ldots \vee L_m^2$, decimos que C_1 y C_2 *se pueden resolver* si existen L_i^1 en C_1 y L_j^2 en C_2 que son *literales opuestos* (uno atómico y el otro su negación). Por ejemplo, P y $\neg P$ son literales opuestos. En este caso, el *resolvente* de C_1 y C_2, es la cláusula:

$$res(C_1, C_2) = L_1^1 \vee \ldots \vee L_{i-1}^1 \vee L_{i+1}^1 \vee \ldots \vee L_n^1 \vee L_1^2 \vee \ldots \vee L_{j-1}^2 \vee L_{j+1}^2 \vee \ldots \vee L_m^2$$

El método de *Resolución* es un mecanismo de demostración por refutación. A partir de un conjunto de cláusulas, trata de conseguir la cláusula vacía (equivalente a falso), lo que demostrará la insatisfactibilidad del conjunto. La idea es la siguiente. El conjunto de cláusulas inicial se va transformando sucesivamente al aplicar un paso de resolución. Un *paso de resolución* consiste en elegir dos cláusulas del conjunto que se puedan resolver y añadir a dicho conjunto el resolvente de ellas.

Primera parte [0,5 puntos]

- Definición de tipos.
  - Define un tipo de datos `Prop` para los símbolos de Σ, que sea instancia de las clases `Show`, `Read` y `Eq` (usando `deriving`).
  - Define un tipo de datos `Formula` (recursivo) para representar las fórmulas permitidas. Tiene que ser instancia de `Read` y `Eq` (usando `deriving`) e instancia de `Show` de manera que se muestre la fórmula de una forma lo más parecida a la notación habitual.
- Definición de funciones para manejar fórmulas y reconocer cláusulas y cláusulas de Horn.
  - Define una función `esClausula` que dada una expresión de tipo `Formula` compruebe si se trata de una cláusula o no.
  - Define una función `fncAlista` que dada una expresión `f` de tipo `Formula`, que está en FNC, devuelve una lista cuyos elementos son las cláusulas que son las componentes de la conjunción `f`. Es decir si $F = C_1 \wedge \ldots \wedge C_n$, el resultado será $[C_1, \ldots, C_n]$.
  - Define una función `clausulaLista` que dada una expresión `c` de tipo `Formula` que es cláusula, devuelva la lista de literales que forman la cláusula. A esta lista la llamaremos *lista de literales asociada a* `c`. Por ejemplo, la lista de literales asociada a la cláusula $P \vee \neg Q \vee S$ es $[P, \neg Q, S]$.
  - Define una función `esClausulaHorn` que dada una expresión de tipo `Formula` que es cláusula, compruebe si se trata de una cláusula de Horn o no.
  - Define una función `resolvente xs1 xs2` que dadas dos expresiones `xs1` y `xs2`, que son listas de literales asociadas a cláusulas que se pueden resolver, devuelva la lista de literales asociada al resolvente de dichas cláusulas.

Segunda parte [0,25 puntos]

Se trata de programar en Haskell la función `resolucion`, especificada más abajo, para comprobar, mediante el método de Resolución, si un conjunto de cláusulas de Horn es insatisfactible, utilizando el siguiente algoritmo:

Se parte de un conjunto de cláusulas de Horn, S, todas ellas con un literal positivo (programa). Y de una cláusula de Horn, G, que no tiene literal positivo (negación del objetivo).
1. Se busca en S una cláusula C que se pueda resolver con G. Si no existe, el algoritmo para, (sin determinar necesariamente la insatisfactibilidad). En caso contrario, se pasa al paso 2.
2. El nuevo G es $res(G, C)$. Si este resolvente es vacío, el algoritmo termina (el conjunto S junto con el G inicial es insatisfactible, $S \models \neg G$). En caso contrario vuelve al paso 1.

En cualquiera de los dos casos de parada, devuelve el G actual. Como puedes comprobar este mecanismo no es un método completo de decisión.

`resolucion xs x = y`, donde `y` es la lista de literales asociada a la cláusula resultante de aplicar el algoritmo anterior a la lista `xs` (que hace el papel del conjunto de cláusulas de Horn S del algoritmo) y a `x` (que hace el papel de la cláusula de Horn G del algoritmo). Tanto las cláusulas de `xs` como `x` vienen dadas mediante las listas de literales asociadas a ellas. Por tanto, `xs` es una lista de listas de literales, mientras que `x` e `y` son listas de literales.
La forma en la que se diseñe la programación de esta función queda a criterio del programador.

Tercera parte [0,25 puntos]

Se trata de programar una pequeña interacción con el usuario, de modo que se pida al usuario que introduzca las fórmulas, se le pregunte en un sencillo menú qué quiere hacer con ellas, de manera que se usen todas las funciones programadas anteriormente y muestre el resultado de lo pedido.
El formato y procedimiento concreto para esta interacción se deja a criterio del programador.

Indicaciones de carácter general a tener en cuenta:

- Hay que declarar los tipos de todas las funciones que se programen, incluidas las funciones auxiliares y locales que pudieran necesitarse.
- Se pueden usar todas las funciones de `Prelude`, es decir, las que se cargan con el sistema, pero no se puede importar ningún otro módulo, lo que quiere decir que, por ejemplo, si se usan operaciones con listas que no están en `Prelude` hay que programarlas.
- Se valorará muy positivamente el uso de funciones de orden superior predefinidas en Haskell y listas intensionales, así como que las funciones recursivas se definan con recursión final.
- El código debe de estar bien documentado mediante comentarios.
- Para poder realizar ejemplos y evaluar la práctica, debe incluirse una colección de fórmulas proposicionales concretas, definidas mediante expresiones de tipo `Formula`.

Entrega de la práctica y calificación:

- La entrega se realizará a través de la tarea abierta del Campus Virtual y consistirá en un solo fichero `.hs` no comprimido. **El nombre del fichero debe ser el nombre y apellidos del autor**.
- Fecha límite para la entrega: **11 de enero de 2023**.[1]
- La nota de la práctica supone el **10 % de la nota final (1 punto)**.

El trabajo es individual. La copia de otros compañeros o de cualquier otra fuente, así como facilitar la copia a otros, será severamente castigado en la calificación **global** de la asignatura. Podéis usar el foro de ayuda comunitaria del CV para formular y contestar preguntas (lo que será valorado).

[1] Se agradece si se entrega antes.