PROGRAMACIÓN DECLARATIVA G. INFORMÁTICA CURSO 2017-18
PRÁCTICA INDIVIDUAL

Objetivo de la práctica: Escribir un programa en Haskell que permita realizar algunas operaciones con fórmulas de la lógica proposicional.

Descripción del problema: Una fórmula de la lógica proposicional está formada por variables proposicionales *p,q,r...* combinadas por conectivos proposicionales: conjunción ($\wedge$), disyunción ($\vee$), negación ($\neg$), implicación ($\rightarrow$) y doble implicación ($\leftrightarrow$). Por ejemplo, las siguientes son fórmulas de la lógica proposicional:

$f_1 \equiv \neg p \rightarrow (p \rightarrow (q \wedge \neg q))$
$f_2 \equiv p \wedge (\neg q \rightarrow \neg p)$
$f_3 \equiv p \wedge q \wedge (\neg q \vee r)$

Para procesar en Haskell fórmulas de esta lógica partimos de las siguientes definiciones de tipos de datos para representar fórmulas:

```
type Var = String  -- nombres de variables
data  FProp =  V Var | No FProp | Y FProp FProp | O FProp FProp | Si FProp FProp | Sii FProp FProp
```

Por dar un ejemplo, la fórmula f_1 se obtendría al evaluar la siguiente función de aridad 0:

```
f1 = Si (No (V "p")) (Si (V "p") (Y (V "q") (No (V "q"))))
```

Queremos disponer de las siguientes funciones (harán falta otras auxiliares, claro):

- `vars`: `vars f` calcula una lista con los nombres de las variables proposicionales que hay en `f`, sin repeticiones (aunque el orden es irrelevante). Por ejemplo, `vars f1` debe evaluarse a `["p","q"]`.
- `tautologia`: reconoce si una fórmula es una tautología o no, es decir, si es cierta para valores de verdad cualesquiera (`True` o `False`) de sus variables proposicionales.
- `satistactible`: reconoce si una fórmula es una satisfactible o no, es decir, si es cierta para algunos valores de verdad de sus variables proposicionales.
- `consecuencia`: reconoce si una fórmula φ_1 es consecuencia lógica de otra φ_2, es decir, si para valores de verdad cualesquiera de las variables proposicionales, cuando φ_2 es cierta φ_1 lo es también.
- `equivalente`: reconoce si dos fórmulas φ_1 y φ_2 son lógicamente equivalentes, es decir, si para valores de verdad cualesquiera de las variables proposicionales, cuando φ_2 es cierta φ_1 lo es también.
- `consecuencias`: dada una lista `fs` de fórmulas, `consecuencias fs` es una lista con cada fórmula `f` de `fs` emparejada con la lista de aquellas fórmulas de `fs` que son consecuencia lógica de `f`.
- `equivalentes`: dada una lista `fs` de fórmulas, `equivalentes fs` es el conjunto cociente de `fs` por la relación de equivalencia lógica, es decir, es una partición de `fs` en sublistas, cada una de las cuales está formada por fórmulas de `fs` equivalentes entre sí.

¿Qué se debe implementar?

Parte básica:

- Programar las funciones anteriores, atendiendo además a las indicaciones que siguen.
  - Hay que declarar los tipos de todas las funciones que se programen, incluidas las funciones auxiliares 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 todas las operaciones con listas que hagan falta y no estén en `Prelude` hay que programarlas.
  - Para poder realizar ejemplos y evaluar la práctica, deben incluirse al menos cinco fórmulas proposicionales concretas (`f1,f2,f3,f4,f5`), definidas mediante funciones de aridad 0 (al estilo de la `f1` de más arriba). Las tres primeras, `f1,f2,f3` deben corresponder a las fórmulas f_1, f_2, f_3 de arriba.
- Declarar `FProp` como instancia de las siguientes clases de tipos:
  - Como instancia de la clase `Eq`, haciendo que la igualdad entre fórmulas coincida con la igualdad estructural (es decir, componente a componente), salvo por el hecho de que el orden en conjunciones o negaciones no importe. Por ejemplo, la fórmula $\neg(p \wedge (p \vee r))$ sería igual a la fórmula $\neg((r \vee p) \wedge p)$.
  - Como instancia de la clase `Ord`, de modo que una fórmula φ sea menor que otra φ' si φ' es consecuencia lógica de φ.
  - Como instancia de la clase `Show` de modo que la visualización de una fórmula sea algo más legible que lo que proporciona directamente `deriving Show`. Por ejemplo, de modo que al evaluar `f1` el resultado se vea como `~p -> (p -> (q /\ ~q))`.

Parte opcional:
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 y muestre el resultado de lo pedido. El formato y procedimiento concreto para esta interacción se deja a criterio del programador.

¿Qué, cómo y cuándo se debe entregar?

- La entrega se realizará a través del Campus Virtual y consistirá en un solo fichero, en el que las explicaciones irán como comentarios Haskell.
- **Fecha límite para la entrega**: **7 de febrero**

¿Cuánto influye la calificación de la práctica en la calificación final?

- La nota de la práctica supone el **10 % de la nota final (1 punto)**.
- Para obtener 0,7 puntos basta dar una solución razonable y bien explicada a la parte básica. La eficiencia no es nuestra mayor preocupación. Para obtener los 0,3 puntos restantes hay que programar la interacción con el usuario.
- 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. Ante las dudas, consultad con el profesor.