Sintaxis de Lógica Proposicional

Laboratorio 1
Lógica para Computación

1 Introducción

El objetivo de este laboratorio es implementar en Haskell algunos conceptos sobre la sintaxis de las proposiciones en Lógica Proposicional.

Recordando como trabaja Haskell, debemos definir un módulo y su nombre debe ser el mismo que el archivo que se está creando:

```
module Lab1 where
```

Tendremos que definir un tipo inductivo **L** para representar las fórmulas de Lógica Proposicional. Es conveniente separar las conectivas binarias agrupándolas todas bajo un solo tipo, como es sugerido en las siguientes reglas de sintaxis:

$$\begin{aligned} \alpha, \beta \ &::= \ p \ | \ (\neg\, \alpha) \ | \ (\alpha \circ \beta) \\ \circ \ &::= \ \wedge \ | \ \vee \ | \ \supset \ | \ \leftrightarrow \end{aligned}$$

Se recomienda también representar las variables como un *alias* del tipo String:

```
type Var = String
```

Ejercicios

1. Completar la definición del tipo de fórmulas (**L**), siguiendo las reglas de sintaxis precedentes.

2. Codificar y nombrar las siguientes fórmulas:

   a. $p \wedge \neg\neg q$

   b. $p \wedge \neg\, q \wedge \neg\, r$

   c. $\neg\neg\, p \vee \neg\, (q \wedge p)$

   d. $\neg(r \supset r) \wedge (\neg\neg\, p \vee \neg\, (q \wedge p))$

3. Definir las siguientes funciones:

   a. $\textsf{\textbf{cantBin :: L}} \rightarrow \textsf{\textbf{Int}}$
   Cuenta la cantidad de conectivas binarias de una fórmula.
   Ejemplo: $\mathsf{cantBin}\ (\neg\neg\ \mathsf{p} \vee \neg\ (\mathsf{q} \wedge \mathsf{p}))\ =\ 2$

   b. $\textsf{\textbf{valores :: L}} \rightarrow [(\textsf{\textbf{Var, Bool}})]$
   Dada una conjunción de *literales* (estos son variables sin negar o variables negadas), devuelve una lista con los nombres de las variables y un valor de verdad asociado. El valor debe ser True si la variable no se encuentra negada y False si se encuentra negada.
   Ejemplo: $\mathsf{valores}\ (\mathsf{p} \wedge \neg\ \mathsf{q} \wedge \neg\ \mathsf{r})\ =\ [(\mathsf{p}, \mathsf{True}), (\mathsf{q}, \mathsf{False}), (\mathsf{r}, \mathsf{False})]$

   c. $\textsf{\textbf{dobleNeg :: L}} \rightarrow \textsf{\textbf{L}}$
   Simplifica una fórmula eliminando las negaciones dobles.
   Ejemplo: $\mathsf{dobleNeg}\ (\neg\neg\ \mathsf{p} \vee \neg\ (\mathsf{q} \wedge \mathsf{p}))\ =\ \mathsf{p} \vee \neg\ (\mathsf{q} \wedge \mathsf{p})$

   d. $\textsf{\textbf{cambiar :: L}} \rightarrow \textsf{\textbf{L}}$
   Sustituye la disyunción ($\vee$) por su equivalente lógico: $\alpha \vee \beta\ \approx\ \neg\ \alpha \supset \beta$
   Ejemplo: $\mathsf{cambiar}\ (\neg\neg\ \mathsf{p} \vee \neg\ (\mathsf{q} \wedge \mathsf{p}))\ =\ \neg\neg\neg\ \mathsf{p} \supset \neg\ (\mathsf{q} \wedge \mathsf{p})$

   e. $\textsf{\textbf{cantPropX :: L}} \rightarrow \textsf{\textbf{Var}} \rightarrow \textsf{\textbf{Int}}$
   Cuenta la cantidad de veces que aparece una letra proposicional en una fórmula.
   Ejemplo: $\mathsf{cantPropX}\ (\neg(\mathsf{r} \supset \mathsf{r}) \wedge (\neg\neg\ \mathsf{p} \vee \neg\ (\mathsf{q} \wedge \mathsf{p})))\ \mathsf{p}\ =\ 2$

   f. $\textsf{\textbf{listarProp :: L}} \rightarrow [\textsf{\textbf{Var}}]$
   Devuelve una lista que contiene las letras proposicionales que aparecen en una fórmula, sin repetidos.
   Ejemplo: $\mathsf{listarProp}\ (\neg(\mathsf{r} \supset \mathsf{r}) \wedge (\neg\neg\ \mathsf{p} \vee \neg\ (\mathsf{q} \wedge \mathsf{p})))\ =\ [\mathsf{p}, \mathsf{q}, \mathsf{r}]$

   g. $\textsf{\textbf{sustCon :: L}} \rightarrow \textsf{\textbf{BC}} \rightarrow \textsf{\textbf{BC}} \rightarrow \textsf{\textbf{L}}$
   Recibe dos conectivas binarias y sustituye la primera por la segunda en una fórmula.
   Ejemplo: $\mathsf{sustCon}\ (\neg\neg\ \mathsf{p} \vee \neg\ (\mathsf{q} \wedge \mathsf{p}))\ \wedge\ \vee =\ \neg\neg\ \mathsf{p} \vee \neg\ (\mathsf{q} \vee \mathsf{p})$

   h. $\textsf{\textbf{swapCon :: L}} \rightarrow \textsf{\textbf{BC}} \rightarrow \textsf{\textbf{BC}} \rightarrow \textsf{\textbf{L}}$
   Idem (g), pero intercambiando ambas conectivas.
   Ejemplo: $\mathsf{swapCon}\ (\neg\neg\ \mathsf{p} \vee \neg\ (\mathsf{q} \wedge \mathsf{p}))\ \wedge\ \vee =\ \neg\neg\ \mathsf{p} \wedge \neg\ (\mathsf{q} \vee \mathsf{p})$

   i. $\textsf{\textbf{invertir :: L}} \rightarrow \textsf{\textbf{L}}$
   Invierte los valores de las variables (ej. p por $\neg$p y $\neg$p por p) y los conectivos de conjunción/disyunción ($\wedge$ por $\vee$ y $\vee$ por $\wedge$). Utilizar dobleNeg para eliminar las dobles negaciones y swapCon para invertir las conectivas binarias.
   Ejemplo: $\mathsf{invertir}\ (\neg\neg\ \mathsf{p} \vee \neg\ (\mathsf{q} \wedge \mathsf{p}))\ =\ \neg\ \mathsf{p} \wedge \neg\ (\neg\mathsf{q} \vee \neg\mathsf{p})$

j. **sustSimp :: Var → L → L → L**
Recibe una variable p, dos fórmulas β y α, y devuelve la fórmula que se obtiene al sustituir p por β cada vez que p aparece en α (esta operación se nota $\alpha[p := \beta]$).
Ejemplo: $\mathsf{sustSimp\ p\ (r \lor s)\ (p \land \neg\neg q)} \ = \ \mathsf{(r \lor s) \land \neg\neg q}$

k. **sustMult :: [(Var, L)] → L → L**
Recibe una sustitución múltiple σ y una fórmula α, y devuelve la fórmula que se obtiene al efectuar la sustitución múltiple σ sobre α (esta operación se nota $\alpha[\sigma]$).
La sustitución múltiple σ es representada por una *lista* de parejas (p, β) donde p es una letra proposicional y β es una fórmula. La idea es que cada pareja (p, β) sirve para indicar que p debe sustituirse por β. Si una letra aparece en dicha lista más de una vez, se considerará solamente su primera aparición, ignorándose las otras.
Ejemplo: $\mathsf{sustMult\ [(p, r \lor s), (q, s \supset t)]\ (p \land \neg\neg q)} \ = \ \mathsf{(r \lor s) \land \neg\neg(s \supset t)}$