Práctica 2
Interpretaciones

Ramírez Rojas José David
Vázquez Cordero Néstor Semer

1 de marzo de 2020

1. Descripción

1.1. Variables

Dado que debemos devolver la lista sin tener variables duplicadas utilizamos la función del lenguaje Haskell *nub* que remueve elementos duplicados de una lista.
Para esto fue necesario importar la librería Data.List
Por casos tratamos los posibles constructores del tipo de dato Prop de tal forma que :

$$Si \quad \varphi \in Var\ P \quad \Rightarrow \quad variables(\varphi) = [\varphi]$$

$$variables(\neg\varphi) = nub(variables(\varphi))$$

$$Si \quad \star \in \{\vee, \wedge, \rightarrow, \leftrightarrow\} \Rightarrow variables(\varphi \star \psi) = nub(variables(\varphi) \quad ++ \quad variables(\psi))$$

1.2. Conjunto Potencia

Sabemos que *conjPotencia* $(\varnothing) = [\{\varnothing\}]$
Luego recursivamente[1] definimos que para una lista *x:xs*, que es un conjunto de cualquier tipo, *conjPotencia (x:xs)* es igual a una lista con la cabeza de la lista junto con todos los otros elementos la lista tales que pertenecen al conjPotencia[1] de la cola(o rabo) de dicha lista, concatenados con la cabeza.

1.3. Interpretación

Volvimos a tratar por casos los posibles constructores del tipo de dato Prop, usando la función nativa de Haskell *elem* que recibe un elemento y una lista y nos indica si dicho elemento forma parte de la lista, haciendo que sea α una lista de estados:

$$Si \quad \varphi \in Var\ P \quad \Rightarrow \quad interpretacion(\varphi, \alpha) = elem(\varphi, \alpha)$$

$$interpretacion(\neg\varphi, \alpha) = \neg(interpretacion(\varphi, \alpha))$$

$$Si \quad \star \in \{\vee, \wedge\} \Rightarrow interpretacion(\varphi \star \psi, \alpha) = interpretacion(\varphi, \alpha) \quad \star \quad interpretacion(\psi, \alpha)$$

$$interpretacion(\varphi \rightarrow \psi, \alpha) = \neg(interpretacion(\varphi, \alpha)) \vee interpretacion(\psi, \alpha)$$

$$interpretacion(\varphi \leftrightarrow \psi, \alpha) = interpretacion(\varphi \rightarrow \psi, \alpha) \wedge interpretacion(\psi \rightarrow \varphi, \alpha)$$

1.4. Estados Posibles

Para cualquier fórmula $\varphi \in Prop$ la lista de estados posibles de dicha fórmula corresponde al *conjPotencia* de la lista de variables que contiene la fórmula.

1.5. Tautología

Para saber si φ es tautología podemos evaluar:

$$tautologia(\varphi) = \bigwedge\{interpretacion(\varphi, l) | l \in estadosPosibles(\varphi)\}$$

1.6. Contradicción

En este problema obtenemos todos los estados de una formula proposicional φ y los negamos, utilizamos el operador and para verificar si hay algun valor falso en la lista, como se negaron todos los estados de φ si es contradiccion al aplicar and el resultado será un true, lo que quiere decir que en realidad todos los valores son falsos, por lo tanto es una contradiccon.

1.7. Es Modelo

Tomamos un estado y una formula proposicional φ tenemos varios casos:

- Cuando φ es T ya que es una constante su valor siempre es True.
- Cuando φ es F igualmente por ser una constante su valor es False.
- Cuando φ es una variable proposicional, si esta pertenece a la lista de estados entonces el resultado es true, en otro caso es falso.
- Cuando $\neg\varphi$ es la negacion de una variable proposicional, el resultado es negar el valor de φ si es que aparece en la lista de estados.
- Cuando tenemos una conjuncion $\varphi \wedge \psi$ y la lista de estados e dividimos la proposicion de la siquiente forma: *EsModelo e* φ && *EsModelo e* ψ, se usa && ya que es el operador logico del haskell.
- Cuando tenemos una disyuncion $\varphi \vee \psi$ y la lista de estados e dividimos la proposicion de la siquiente forma: *EsModelo e* φ || *EsModelo e* ψ, se usa || ya que es el operador logico del haskell.
- Cuando tenemos una implicacion $\varphi \Rightarrow \psi$ y la lista de estados e pasamos a su forma normal negativa y dividimos la proposicion de la siquiente forma: *EsModelo e* $\neg\varphi$ || *EsModelo e* ψ.

- Cuando tenemos una equivalencia $\varphi \Rightarrow \psi$ y la lista de estados e lo separamos a una doble implicacion de la siquiente forma: $(\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$, entonces resolvemos recursivamente con la funcion: *EsModelo e* $\varphi \Rightarrow$ *EsModelo e* ψ && *EsModelo e* $\varphi \Rightarrow$

1.8. Modelos

Lo que hacemos en este problema es obtener primero todos los etados posibles de una formula proposicional φ, luego,usamos la funcion que nos dice si φ es un modelo, si lo es entonces se agrega a la lista, al finalizar obtenemos una lista con todos los modelos de una formula proposicional φ.

1.9. Es Insatisfacible

Para saber si una formula φ es insatisfacible utilizamos la funcion contradiccion, si encontramos que la formula φ es contradicción significa que no existe interpretacion que haga satisfacible a la funcion, por lo tanto si la duncion contradiccion nos retorna True es porque no existe un modelos para φ.

1.10. Es Satisfacible

Para saber si una formula φ es insatisfacible utilizamos la funcion contradiccion, si encontramos que la formula φ es contradicción significa que no existe interpretacion que haga satisfacible a la funcion, por lo tanto si la duncion contradiccion nos retorna True es porque no existe un modelos para φ es satisfacible.

2. Forma de ejecución

Esta práctica se puede ejecutar usando el ambiente del compilador GHC: GHCi, escribiendo en la terminal lo siguiente:
:∼$ ghci NestorVazquez/Practica2/src/Practica2.hs
*Hay que tener cuidado con el uso de los parentesis al utilizar cualquier función.

3. Conclusiones

En esta práctica se revisaron mecanismos para verificar si una fórmula escrita en lógica proposicional es correcta además aprendimos a trabajar con un tipo de dato personalizado que tiene varios constructores.