Lógica Computacional: Practica 3

José Miguel Toledo Reyes
Omar Fernando Gramer Muñoz

17 de marzo de 2020

1. Realización

La practica se realizó en conjunto entre los dos miembros del equipo, esto con el fin de estandarizar el código y mantener una mejor coherencia en el formato.

Para resolver los problemas se decidió optar por una estrategia de divide y vencerás, esto quiere decir que para cada uno de los problemas de la práctica decidimos analizamos de tal manera que pudieramos descomponerlo (en caso de ser necesario) en funciones más simples (funciones auxiliares), dependiendo de la naturaleza de cada problema.
Esta técnica es más evidente en la implementación de las funciones del archivo `mDPLL.hs`

Además cabe mencionar que hubieron dos funciones en particular del archivo `LProp.hs` de las cuales tuvimos que hacer un análisis especial en como haríamos su implementación para poder realizarlas.

- La primera de ellas es la función `meteNeg`, la cual pide distribuir negaciones de una funcíon hasta que llegen a las fórmulas atómicas que la componen.

  Uno podría pensar que basta con hacer una recursión estructural para cada una de las definiciones que componen a una fórmula propocisional y aplicar leyes de DeMorgan donde sea necesario, sin embargo esto no es tan simple, ya que en la implementación de proposiciones que tenemos en la práctica no podemos saber (al menos no de manera directa) cómo es el interior de la negación de una proposición, es decir, que si recibimos como parametro `(Neg Φ)`, donde Φ es una fórmula proposicional ,entonces no podemos saber si Φ es una variable, una disyunción, una constante, etc.

  Ante el problema dado, decidimos crear una función `auxNeg` la cual emula una "sobrecarga en la pila"de las negaciones. Así podemos recibir fórmulas sin una construcción del tipo `(Neg Φ)` para poder procesarlas de acuerdo al conectivo lógico de su construcción, para esto es importante tomar en cuenta que si se hizo una llamada a esta funcien auxiliar es porque la fórmula que recibimos estaba negada, y por lo tanto es necesario procesarla como tal.

Es decir, que si se recibe una formula del tipo `(Disy Φ Ψ )` entonces hay que contemplar que dicha fórmula estaba negada antes de haber sido recibida como parámetro y por lo tanto se debe aplicar una recursión con leyes de DeMorgan. Del mismo modo si se recibe una fórmula `(Neg Φ)` eso implica que la fórmula tenía una doble negación antes de haber sido recibida como parámetro y por lo tanto debemos cancelar el par de negaciones al hacer recursión.

- La segunda función fue `dist`, la cual distribuye disyunciones de la fórmula sobre sus conjunciones.
  Para esta función se hizo un análisis semejante al de la función anterior `meteNeg`, es decir, que al implementarla decidimos hacer una función auxiliar que se apoyara sobre la pila de llamadas para simular la carga de las disyunciones que fueramos encontrando. En esta implemetación hay que además verificar con otra función auxilar si alguna de las subfórmulas en la cuales se hará recursión contiene conjunciones ya que de lo contrario podemos crear un bucle infinito y nunca terminar de procesar.

Para implementar las funciones del archivo `DPLL.hs` simplemente nos basamos en las definiciones de cada una de las reglas que conforman al algoritmo, las cuales fueron proporcionadas en las notas de la materia. Además para facilitar la implementación de las funciones fue necesario utilizar listas por comprensión y, como se había dicho al comienzo, optar por una estrategia de divide y vencerás.

Además cabe mencionar que la implementación que proporcionamos para las funciones de DPLL utilizan todas las variables posibles con las cuales se pueda trabajar en una sola llamada. Es decir, que si por citar un ejemplo se hace uso de la función `unit` entonces se agregarán al modelo todas las clausulas unitarias que se encuentren en la fórmula.
La única función que debido a su complejidad no fué implementada para trabajar con todas las variables posibles a la vez fue `split`.

2. Ejecución del programa

Para poder compilar el programa es necesario tener instalado GHC compiler, el cual puede obtenerse mediante el siguiente comando en Linux.

```
sudo apt-get install haskell-platform
```

Una vez hecho lo anterior, para correr el programa, es necesario establecerse el directorio de trabajo en la carpeta en donde se encuentran los archivos LProp.hs y DPLL.hs, y ejecutar los siguientes comandos:

- Ejecución del archivo LProp.hs:

  ```
  ghci LProp.hs
  ```

  Esto hará que se compile el programa y sea posible su ejecución.
  Para ejecutar cualquiera de las funciones del programa solo basta escribir:

  ```
  *LProp>  [Nombre de la función] [parámetro] [parámetro] ...[parámetro]
  ```

- Ejecución DPLL:

  ```
  ghci DPLL.hs
  ```

  Esto hará que se compile el programa y sea posible su ejecución.Cabe resaltar que para que se pueda compilar el archivo, se debe de contar con que el archivo LProp.hs compile de manera correcta, ya que el archivo DPLL.hs,depende de cierta manera del archivo antes mencionado para su funcionamiento.

  Asi mismo al igual que con el anterior archivo,para ejecutar cualquiera de las funciones del programa solo basta escribir:

  ```
  *DPLL>  [Nombre de la función] [parámetro] [parámetro] ...[parámetro]
  ```

3. Conclusiones

Esta práctica nos ayudó a conocer mas a fondo las características del lenguaje de haskell como lo fueron:

El manejo de listas por comprensión, ya que tuvimos que aprender las distintas maneras el la que se podian obtener distintos tipos de datos atraves de ellas,cosa que nos fue de mucha ayuda al momento de implementar las funciones del archivo DPLL.hs,las cuales en su mayoria era una buena alternativa para hacer mucho mas legible y compacta la manera en la que se iba a implementar la función y evitar así tambien la implementación innecesaria de funciones auxiliares.

Asi mismo tambien conocimos mas a fondo la declaración de tipos de datos en haskell,cosa con la que fue posible abstraer y manejar las propociones logicas de una manera similar a

como lo hacemos en clase y asi mismo hacer posible la implementacion de las distintas reglas y procedimientos necesarios para poder llevar acabo el algoritmo DPLL.