Facultad de Ciencias - UNAM

Lógica Computacional 2020-2

Práctica 2: SAT y el algoritmo DPLL.

Parte 1

Favio E. Miranda Perea
Alejandra Krystel Coloapa Díaz
Pedro Juan Salvador Sánchez Pérez

28 de febrero de 2020
fecha de entrega: 12/03/2020

Haciendo uso del algoritmo DPLL de la nota 4, el objetivo es poder resolver el problema de satisfacibilidad para la lógica proposicional. Hay que recordar que las fórmulas proposicionales deben de estar en forma nomal conjuntiva y por eso es necesario terminar el archivo LProp.hs

En esta primera parte se centraran a en implementar las reglas del algoritmo DPLL y en fortalecer su conocimiento sobre las listas por comprensión, ya que sera vital su uso para que puedan implementar las reglas fácil y entendibles.

Para representar de manera más cómoda las fórmulas, cláusulas y modelos se crearon los siguientes tipos de datos:

```
type Literal = Prop
type Clausula = [Literal]
type Formula = [Clausula]
type Modelo = [Literal]
type Solucion = (Modelo, Formula)
```

1. Ejercicios

1. (1 punto) Implementa las funciones `elimEquiv, elimImp, meteNeg y dist` vistas en clase para que pueda funcionar bien la función `cnf` .
   Ejemplos
   - `Practica1*>cnf (Imp (V "p") (Conj (V r") (V "q")))`
     `(( ∼ ("p") ∨ r") ∧ ( ∼ ("p") ∨ "q"))`
2. (1.5 puntos) Implementa la función `unit` que dada una Solucion regresa otra Solucion siguiendo la regla de la cláusula unitaria, si es posible, en otro caso regresa la Solucion

de entrada. Se recomienda hacer uso de una o dos funciones auxiliares para alcanzar un código legible.

Ejemplos

- `DPLL>unit ([], [[V "p", V "q"],[V "p"], [Neg (V r")], [V r", V "q"]])`
  `(["p"],[["p","q"],[~(r")],[r","q"]])`
- `DPLL>unit ([], [[V "p", V "q"], [Neg (V r")], [V r", V "q"]])`
  `([~(r")],[["p","q"],[r","q"]])`
- `DPLL>unit ([], [[V "p", V "q"],[V "p", V r"], [V r", V "q"]])`
  `([],[["p","q"],["p",r"],[r","q"]])`

3. (1.5 puntos) Implementa la función `elim` que dada una Solucion regresa otra Solucion siguiendo la regla de la eliminación, si es posible, en otro caso regresa la Solucion de entrada. Se recomienda hacer uso de una o dos funciones auxiliares para alcanzar un código legible.

   Ejemplos

   - `DPLL>elim ([V "p"], [[V "p", V "q"], [V r"], [V r", Neg (V "p")], [V r", V "s", V "p"]])`
     `(["p"],[[r"],[r",~("p")]])`
   - `DPLL>elim ([Neg (V "p")], [[V "p", V "q"], [V r"], [V r", Neg (V "p")], [V r", V "s", V "p"]])`
     `([~("p")],[["p","q"],[r"],[r","s","p"]])`
   - `DPLL>elim ([Neg (V r")], [[V "p", V "q"], [V r"], [V r", Neg (V "p")], [V r", V "s", V "p"]])`
     `([~(r")],[["p","q"],[r"],[r",~("p")],[r","s","p"]])`

4. (1.5 puntos) Implementa la función `red` que dada una Solucion regresa otra Solucion siguiendo la regla de la reducción, si es posible, en otro caso regresa la Solucion de entrada. Se recomienda hacer uso de una o dos funciones auxiliares para alcanzar un código legible.

   Ejemplos

   - `DPLL>red ([V "p"],[[V "p", V "q"], [Neg (V "p"), V r"], [V "q", Neg (V r"), Neg (V "p"), V "s"]])`
     `(["p"],[["p","q"],[r"],["q",~(r"),"s"]])`
   - `DPLL>red ([Neg (V "p")],[[V "p", V "q"], [Neg (V "p"), V r"], [V "q", Neg (V r"), Neg (V "p"), V "s"]])`
     `([~("p")],[["q"],[~("p"),r"],["q",~(r"),~("p"),"s"]])`
   - `DPLL>red ([V "q"],[[V "p", V "q"], [Neg (V "p"), V r"], [V "q", Neg (V r"), Neg (V "p"), V "s"]])`
     `(["q"],[["p","q"],[~("p"),r"],["q",~(r"),~("p"),"s"]])`

5. (1.5 puntos) Implementa la función `split` que dada una Solucion regresa una lista de tipo Solucion siguiendo la regla de la separación, si es posible, en otro caso regresa la Solucion de entrada dentro de una lista. Se recomienda hacer uso de una o dos funciones auxiliares para alcanzar un código legible.

   Ejemplos

- `DPLL>split ([],[[V "p", V "q"], [Neg (V "p"), V r"], [V "q", Neg (V Γ "), Neg (V "p"), V "s"]])`
  `[(["p"],[["p","q"],[∼("p"),r"],["q",∼(r"),∼("p"),"s"]]),`
  `([∼("p")],[["p","q"],[∼("p"),r"],["q",∼(r"),∼("p"),"s"]])]`
- `DPLL>split ([V "p"],[[V "p", V "q"], [Neg (V "p"), V r"], [V "q", Neg (V r"), Neg (V "p"), V "s"]])`
  `[([∼(r"),"p"],[["p","q"],[∼("p"),r"],["q",∼(r"),∼("p"),"s"]]),`
  `([r","p"],[["p","q"],[∼("p"),r"],["q",∼(r"),∼("p"),"s"]])]`
- `DPLL>split ([V "p", V r", V "q", V "s"],[[V "p", V "q"], [Neg (V "p"), V r"], [V "q", Neg (V r"), Neg (V "p"), V "s"]])`
  `[(["p",r","q","s"],[["p","q"],[∼("p"),r"],["q",∼(r"),∼("p"),"s"]])]`

6. (.25 puntos) Implementa la función `conlfict` siguiendo la regla de conflicto, que para una Solucion dada regresa True si la cláusula vacia se encuentra en la fórmula, regresa False en otro caso.
   Ejemplos
   - `DPLL>conflict ([V "p"], [[]])`
     `True`
   - `DPLL>conflict ([V "p"], [[V "q"]])`
     `False`
7. (.25 puntos) Implementa la función `success` siguiendo la regla de éxito, que para una Solucion dada regresa True si la fórmula no contiene cláusulas, regresa False en otro caso.
   Ejemplos
   - `DPLL>success ([V "p"], [])`
     `True`
   - `DPLL>false ([V "p"], [[V "q"]])`
     `False`
8. (2.5 puntos) Implementar todas las funciones del archivo LConj. Es la representacion de la estructura de datos "Conjunto" implementado con Listas de haskell. Es fundamental que se implementen todas las funciones con el mecanismo de "list comprehension"de haskell de lo contrario no se evaluarán las funciones.

2. Puntos extra

1. (3 puntos) Implementa las funciones de la seccion 3.1 de la nota 2 del curso correspondiente a la semantica de la lógica proposicional, usa el archivo LConj cuando lo necesites en lugar de funciones de Haskell que hagan lo mismo.

3. Sugerencias generales

- hacer uso solo de la nota 4 del curso ya que otras fuentes en internet podrían confundir demasiado.

- Hacer uso de funciones auxiliares y sobre todo de list comprehension ya que estas reducirán el código y lo harán completamente legible y es mas fácil de entender lo que se está haciendo ya que se aplica muy directamente lo que se desea hacer, facilita mucho la implementación de la práctica en especial para el algoritmo dpll.