Resolución

November 16, 2021

Contents

1 Pasos **1**

2 Formas normales **2**
2.1 Forma normal negativa 2
2.1.1 Pasos 2
2.2 Forma normal conjuntiva 3
2.3 Forma clausular 3

3 Resolución **3**
3.1 Resolver cláusulas 3
3.1.1 Pasos 4
3.2 Satisfacibilidad de clausulas 4
3.2.1 Triviales 4
3.2.2 Insatifacibles 4
3.2.3 Lista de cláusulas 5
3.3 SAT para fórmula arbitraria 5
3.3.1 Se tiene la fórmula como lista de cláusulas 5
3.3.2 Parar si todas las clausulas están resueltas 5
3.3.3 Clausas complemetarias que no haya sido escojida antes 5
3.3.4 Obtener la resolución de esas cláusulas 5
3.3.5 Si no es trivial, se agrega a la lista de cláusulas 6
3.3.6 Si la clausula es vacía, deternerse y fallar 6

1 Pasos

1. Pasar a forma normal conjuntiva
2. Pasar a forma clausular

3. Resolver clausulas hasta
   (a) Resolver todos los conflictos
   (b) Encontrar la cláusula vacía

2 Formas normales

- Tranformar una fórmula a otra equivalente con una estructura más simple

2.1 Forma normal negativa

- Solo variables, negaciones de variables, conjunciones y disjunciones

2.1.1 Pasos

1. Eliminar implicaciones y equivalencias

$$p \implies q \to \neg p \vee q$$

$$p \iff q \to \neg p \vee q \wedge \neg q \vee p$$

1. Meter negaciones (leyes de Morgan)

$$\neg(p \wedge q) \to \neg p \vee \neg q$$

$$\neg(p \vee q) \to \neg p \wedge \neg q$$

- En Haskell

```
toNnf :: Formula a -> Formula a
```

2.2 Forma normal conjuntiva

- Forma normal negativa + conjunción de disjunciones
- Leyes distributivas para la disjunción

$$p \vee (q \wedge r) \to (p \vee q) \wedge (p \vee r)$$

- En Haskell

```
toCnf :: Formula a -> Formula a
```

2.3 Forma clausular

- Notación para la forma normal conjuntiva

$$(p \vee q \vee r) \wedge (\neg r \vee p) \wedge (\neg s \vee q) \to \{pqr, \bar{r}p, \bar{s}q\}$$

- En Haskell

```
data Literal a = N a | P a

type Clause a = [Literal a]

type ClausulalForm a = [Clausulal a]
```

- Y para cambiar de notación

```
toClausulal :: Formula a -> ClausulalForm a
```

3 Resolución

3.1 Resolver cláusulas

- Dadas dos cláusulas, ¿qué es necesario para que ambas sean verdaderas?

3.1.1 Pasos

1. Seleccionar dos cláusulas c_1, c_2 con una literal complementaria, digamos l
2. Quitar la literal complementaria de ambas clausulas
3. Juntar las dos clausulas en una

Sin perdida de generalidad, esto se puede expresar de la siguiente manera

$$c' = (c_1 - l) + (c_2 - l^c)$$

- Para Haskell

```
resolve :: Clause a -> Clause a -> Clause a
```

3.2 Satisfacibilidad de clausulas

- Clausula $\rightarrow \bigvee$
- Es verdadera si alguna de sus literales es verdadera

3.2.1 Triviales

- Si tiene una tautología es verdadera
- $p \vee \neg p \rightarrow \{p\bar{p}\}$ es tautología
- Si una clausula contiene literales complemetarias, es trivialmente verdadera

```
clashes :: Clause a -> Clause a -> Maybe a

isTrivial :: Clause a -> Clause a
isTrivial c = isJust $ clashes c c
```

3.2.2 Insatifacibles

- Si está vacía, no tiene literales
- Como no puede tener una literal verdadera, es falsa
- La clausula vacía $\square$ es insatisfacible

```haskell
isEmpty :: Clause a -> Bool
isEmpty [] = True
isEmpty _ = False
```

3.2.3 Lista de cláusulas

- Lista de clausulas $\rightarrow \bigwedge$
- Es falsa si alguna de sus literales es falsa
- Si una forma clausular tiene una clausula vacía, es insatisfacible

```haskell
hasEmpty :: ClausulalForm a -> Bool
```

3.3 SAT para fórmula arbitraria

3.3.1 Se tiene la fórmula como lista de cláusulas

```haskell
resolution :: ClausulalForm a -> Bool
```

3.3.2 Parar si todas las clausulas están resueltas

```haskell
clashList :: ClausulalForm a -> [(Clause a, Clause a)]

resolution cs =
  | null $ clashList cs = True
```

3.3.3 Clausas complemetarias que no haya sido escojida antes

- Hay que mantener un registro de las clausuas que han sido usadas :(

```haskell
data ResolReg a = ResolReg (ClausulalForm a) [(Clause a, Clause a)]

someClash :: ResolReg a -> Maybe (Clause a, Clause a)
```

3.3.4 Obtener la resolución de esas cláusulas

```haskell
resolution cs
  | null clashList cs = True
  | otherwise = applyRes $ ResolReg (not . isTrivial $ cs) []

applyRes :: ResolReg a => Bool
applyRes r@(ResolReg cs rs) =
```

```
  case someClash r of
    Nothing -> True
    Just (c1, c2) -> let c = resolve c1 c2 in undefined
```

3.3.5 Si no es trivial, se agrega a la lista de cláusulas

```
Just (c1, c2) -> let c = resolve c1 c2 in
  let clauses = if isTrivial c then cs else (c:cs) in
```

3.3.6 Si la clausula es vacía, deternerse y fallar

```
if isEmpty c then False else applyRes clauses (c1, c2):rs
```