This is my propostional parser using Parsec (parsec2 package)

```
module PropParser where

import Text.ParserCombinators.Parsec -- :set -ignore-package parsec-3.1.0
import Text.ParserCombinators.Parsec.Expr
import qualified Text.ParserCombinators.Parsec.Token as T
import Text.ParserCombinators.Parsec.Language (haskellDef)
import Text.ParserCombinators.Parsec.Char

import Myitautology
import Char

-- The Parser
mainParser = do whiteSpace
                e <- expr
                eof
                return e

expr    = buildExpressionParser table term
        <?> "expression"

term0    =  parens expr
        <|> var
        <?> "simple proposition"

term     = do
        t <- term0
        whiteSpace
        return t

table   :: OperatorTable Char () Prop
table   = [ [prefix "~" Not ]
          , [binary "&" And AssocLeft, binary "v" Or AssocLeft ]
          , [binary "=>" Imply AssocLeft, binary "<=>" Equiv AssocNone ]
          ]

binary  name fun assoc = Infix (do{ reservedOp name; whiteSpace; return fun }) assoc
prefix  name fun       = Prefix (do{ reservedOp name; whiteSpace; return fun })
postfix name fun       = Postfix (do{ reservedOp name; whiteSpace; return fun })

isVar                   :: Char -> Bool
isVar c                 =  isAlpha c && c /= 'v'

var                     :: Parser Prop
var                     =  fmap Var $ satisfy isVar
```

```
evalProp               :: String -> String -> Rests -> String
evalProp x1 x2 rs
                =  case (parse mainParser "" x1) of
                        Left err1 -> show err1
                        Right  p1 -> case (parse mainParser "" x2) of
                                        Left err2 -> show err2
                                        Right  p2 -> show (propMachine p1 p2 rs)

-- This will convert the first string to a Prop and allow disagree to work
evalDisagree            :: String -> String -> Rests -> String
evalDisagree x1 x2 rs    =  case (parse mainParser "" x1) of
                                Left err1 -> show err1
                                Right p1 -> case (parse mainParser "" x2) of
                                                Left err2 -> show err2
                                                Right  p2 -> disagree p1 p2 rs

-- Restriction parser
getRests                :: [String] -> Rests -> Either String Rests
getRests []     ps      =  Right ps
getRests (r:rs) ps      =  case (parse mainParser "" r) of
                                Left err -> Left "hello"
                                Right  p -> Right [p]

removeEmpty xs = filter (/= "") xs

-- The lexer
lexer      = T.makeTokenParser haskellDef
lexeme     = T.lexeme

parens     = T.parens lexer
natural    = T.natural lexer
reservedOp  = T.reservedOp lexer
whiteSpace  = T.whiteSpace lexer
```