```
(*Problem 3.1: A datatype declaration that can be used to represent
 logical expressions *)

datatype expr = var of string | AND of expr * expr
 | OR of expr * expr | NOT of expr;

(*Problem 3.2:
 The assignment of truth values for propositional variables within a logical
 expression can be represented as a list of tuples containing the name of the
 variable and the truth assignment for it. Thus, this representation would have
 type (string*bool) list. As an example, [("p, true"), ("q", true)] is an
 assignment list whereby the propositional variables p and q within some given
 expression both have the value true.*)

(*Problem 3.3 A function that returns the truth value of a logical Expression E
 and an assignment L of truth values for propositional variables. This function
 makes use of a helper function truthValue that identifies the boolean value of
  a specific propositional variable.*)

fun truthValue [] prop = false
 | truthValue ((name, value)::rest) prop =
  if name = prop then value else truthValue rest prop
;

fun eval (List, (var x)) = truthValue List x
 | eval (List, (AND(left, right))) =
  if (eval (List, left)) = true then
   if (eval (List, right)) = true then
    true
   else false
  else false
 | eval (List, (OR(left, right))) =
  if (eval (List, left)) = false then
   if (eval (List, right)) = false then
    false
   else true
  else true
 | eval (List, (NOT(value))) =
  if (eval (List, value)) = true then
   false
  else true
;

(*Problem 3.4 A function that takes a logical expression and returns a list of
 all the propositional variables appearing in that list. *)

 fun removeDupes [] str = str::[]
  | removeDupes (head::rest) str =
   if head = str then removeDupes rest str else head::(removeDupes rest str)
 ;

fun varsInExp (List, (var x)) = removeDupes List x
 | varsInExp (List, (AND(left, right))) =
```

```
  let val x = (varsInExp (List, left))
  in
  (varsInExp (x, right))
  end
 | varsInExp (List, (OR(left, right))) =
  let val x = (varsInExp (List, left))
  in
  (varsInExp (x, right))
  end
 | varsInExp (List, (NOT(value))) =
  (varsInExp (List, value))
;

(*Problem 3.5 A function that takes a logical expression as argument and
 returns true if the expression is a tautology and false otherwise. Uses 2
 helper functions: combine creates an assignment list from a list of propositional
 variables, while flipValue changes the truth assignments for the variables in
 the assignment list. Raises an END exception if no new truth assignments can
 occur*)

exception END

 fun flipValue [] = raise END
  | flipValue ((name, value)::rest) =
   if value = false then (name, true)::rest else (name, value)::(flipValue rest)
 ;

(*Function initially sets all assignments to false*)
fun combine [] value = []
 | combine (head::rest) value =
   (head, value)::(combine rest value)
;

fun isTaut (expr) =
 let
  fun flipValue2 assign = (flipValue assign) handle END => [("END", false)]
  fun checker L E =
   if eval(L, E) = true then
    let val newL = flipValue2 L
    in
    if newL = [("END", false)] then true else checker newL E
    end
   else false
  val varList = varsInExp([], expr)
  val inititalAssignemnt = combine varList false
 in
  checker inititalAssignemnt expr
 end
;


Control.Print.printDepth := 10;
Control.Print.printLength := 10;

(*TESTS*)
val expresion1 = AND(OR(var("p"), var("q")),NOT(var("p")));
```

```
110 val assignment1 = [("p", false), ("q", true)];
111
112 (*Testing eval function - should return true*)
113 val expression1Eval = eval(assignment1, expresion1);
114
115 (*Testing varsInExp function - should return true*)
116 val allVars = varsInExp([], expresion1);
117
118 (*Testing isTaut function*)
119 val expresion2 = OR(var("p"), NOT(var("p")));
120 val isExprTaut = isTaut(expresion2);
```
