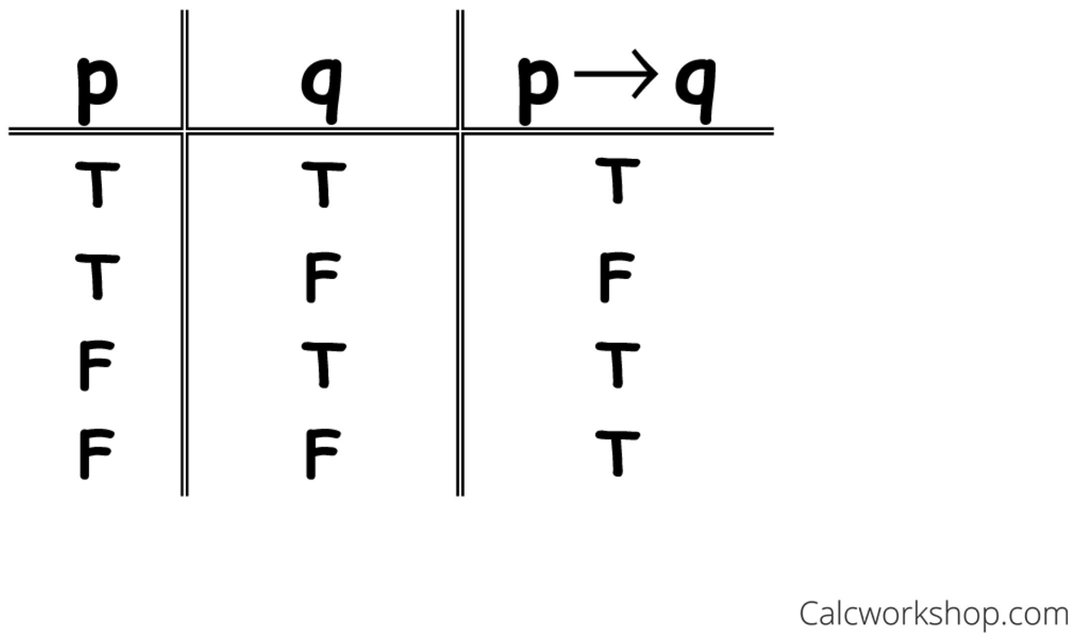
Mandatory Assignment 5

Name: Bjørnar Haugstad Jåtten
ITU-username: bjja@itu.dk

Task 1

To negate a ROBDD u, I use one of the Boolean operators and a terminal ROBDD (0 or 1) as my other argument to *APPLY*. When choosing one of the 16 Boolean operators, we would first have to choose one that had a dual input system, so we both could input the terminal ROBDD and a u. Such an operator would be: AND, OR, NOR or IMPLICATION.

While looking at the truth tables, I found that implication returns the oposite value for input p when input q is 0:



p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Calcworkshop.com

When I then give the operator -> as an input to *APPLY*, together with the terminal ROBDD 0, the given ROBDD u would be negated:

```
APPLY(->, u, 0)
```

Task 2

When we want to construct the ROBDD for a variable x_i, we start of by calling the make function with the variable, and a high and low value for the given variable. A requirement for this function is that the index for the low and the high, are indexes that are already in the unique table.

Before creating the variable, the make-function will test for two requirements:

1. If the test is redundant.
    - This is done by checking if the low and the high of the variable is equal to eachother. If this is true, the test is redundant, and we can return either the low or the high, as they would be the same.
2. If the test is unique.
    - To see if the test is unique, we use the function member, and check if ROBDD we want to create is to be found in the unique table H. If the test is not unique, we return the already existing ROBDD instead of creating a new one.

If both of these checks fail, we will create the ROBDD. First the make-function will have to use the add function to get the next non-occupied ID from the table T. Using this non-occupied ID, we would insert the ROBDD into the unique table H, and return the ID.

In this way, we would have constructed an ROBDD for a variable x_i

Task 3

If the expresion type is VAR we create a new node, with a call to the make-function. The way that this function constructs a ROBDD for a variable is explained in task 2. As the high and the low of value, we give the left and the right expression node of the expression. Using *expr.idx()* we are able to get the ID of the new node.

If the expression is either true or false, we return 1 or 0 accordingly to represent the boolean values.

If the expression type is NOT, we call apply in the way explained in task 1. Our ROBDD u is in this case expr.right(), as the assignment stated that we should assume that it is the right value that is under evaluation.

If the expression is either AND or OR, we use Apply with their according boolean expression. and recursivly call EXPR2ROBDD.

We are always evalutating the right and left expressions of expr to get to an end value. This is done by a recursive call to EXPR2ROBDD.

```
EXPR2ROBDD(expr)
if expr.type() == VAR then
   return Mk(expr.idx(), EXPR2ROBDD(expr.left()), EXPR2ROBDD(expr.right()))
else if expr.type() == TRUE then
   return 1
else if expr.type() == FALSE then
   return 0
else if expr.type() == NOT then
   return Apply(->, EXPR2ROBDD(expr.right()), 0)
else if expr.type() == AND then
   return Apply(^, EXPR2ROBDD(expr.left()), EXPR2ROBDD(expr.right()))
else if expr.type() == OR then
   return Apply(v, EXPR2ROBDD(expr.left()), EXPR2ROBDD(expr.right()))
```