```
'The code below can be used to evaluate the performance of systems O, P, Z, and QC in reasoning about an uncertain envirnoment.
'
'The program operates over an extension of propositional language, with four atomic proposition (a, b, c, and d),
'and the connectives "+" for "and", "|" for "or", ">" for the material conditional, "~" for "not", and "-" for an uncertain conditional.
'(Note that some parts of the code "|" is also used in representing conditional probabilities of the form "P|Q".)
'
'To run the program simply:
'(1) Download, install, and run Microsoft Visual Basic Express Edition or a comperable Microsoft product.
'(2) Select "New Project" and create a "Windows From Aplication" (choose any name).
'(3) Select the "View" "Code" for the project's default form,
'(4) Paste all of the text of the current document over the default code for Form1.
'(5) Select the option "Start Debugging".
'
'After steps (1) through (5) have been completed, the program will excute, and generate a text file "data2.txt" at location C:\ (and also a text file "data1.txt" if Number_of_Runs
= 1).
'The data recorded in the file data2.txt is self explanatory.
'(In the case where Number_of_Runs = 1, data1.txt lists all of the derived conditionals, for each system, along with the derived probabilities (above) and the true probabilities
(below).
'
'Many parameters for the program are inputed by the user. Lines where the user may edit crucial values are highlighed in yellow (on pgs 1 to 3). Default values are currently set.

Public Class Form1
    Dim Proposition(95) As String 'stores ´propositions' corrsponding to the fundamental probability distribution and for all possible conjuctions of literals
    Dim Probability(95) As Double 'stores the true probabilities for the above propositions

    Dim RandomClass As New Random()
    Dim RandomNumber As Double

    Dim Datum As String 'used to store a random queried proposition
    Dim Datum_Prob As Double 'stores the probability of above proposition

    Dim Number_of_Runs As Integer = 1 'number of times the main loop executes
    Dim Number_of_Base_Conditionals As Integer = 2 '******stores number of clues given before first round ******
    Dim Number_of_User_Inputed_Base_Conditionals As Integer = 0 ' This value MUST BE set to the number of user inputed base conditionals.
    Dim Min_Prob_for_Base_Conds As Double = 0.9 'this sets the minimal probability for clues
    Dim Scenario As Integer = 0 'User may set to 1 or 2.

    Dim Degree_of_Normicity As Integer = 0
    Dim High_Value As Double = 0.95
    Dim Low_Value As Double = 0.05

    Dim Clue_PPropositions(Number_of_Base_Conditionals) As String  'these 3 arrays store all information for propositions given as premises
    Dim Clue_Propositions(Number_of_Base_Conditionals) As String
    Dim Clue_Probabilities(Number_of_Base_Conditionals) As Double

    Dim Number_of_Props_Queried As Integer = 464 'full set = 464  ***************************************************************************************************

    Dim Queried_PPropositions(Number_of_Props_Queried) As String  'these 3 arrays store all information for propositions to be checked as consequences for the 3 systems
    Dim Queried_Probabilities(Number_of_Props_Queried) As Double
    Dim Queried_Propositions(Number_of_Props_Queried) As String

    Dim Z_Judgments(Number_of_Props_Queried) As Integer  'these variables record the results of queries by System Z
    Dim Z_LBounds(Number_of_Props_Queried) As Double
    Dim Z_Conclusions(Number_of_Props_Queried) As String

    Dim Zs_Judgments(Number_of_Props_Queried) As Integer  'these variables record the results of queries by System Z
    Dim Zs_LBounds(Number_of_Props_Queried) As Double
    Dim Zs_Conclusions(Number_of_Props_Queried) As String

    Dim P_Judgments(Number_of_Props_Queried) As Integer  'these variables record the results of queries by System P
    Dim P_LBounds(Number_of_Props_Queried) As Double
    Dim P_Conclusions(Number_of_Props_Queried) As String
    Dim P_Library(Number_of_Base_Conditionals + 1, 2 * (2 ^ (Number_of_Base_Conditionals + 1))) As String
    Dim P_Library_Test(2 * 2 ^ (Number_of_Base_Conditionals + 1)) As Integer
    Dim P_Library_Count As Integer = 1

    Dim Classical_Judgments(Number_of_Props_Queried) As Integer  'these variables record the results of queries by classical logic
    Dim Classical_LBounds(Number_of_Props_Queried) As Double
    Dim Classical_Conclusions(Number_of_Props_Queried) As String

    Dim O_LBounds(464) As Double 'these variables record the results of queries by System O
    Dim O_Conclusions(464) As String

    Dim Initial_Size_of_Set As Integer = Number_of_Base_Conditionals
    Dim Size_of_Set As Integer = Number_of_Base_Conditionals
    Dim D_Size_of_Set As Integer = Number_of_Base_Conditionals
    Dim S_Size_of_Set As Integer

    Dim Power_Set1(Number_of_Base_Conditionals, 2 ^ (Number_of_Base_Conditionals)) As String
    Dim Power_Set2(Number_of_Base_Conditionals, 2 ^ (Number_of_Base_Conditionals)) As Double
    Dim Power_Set_Bound_sums(2 ^ (Number_of_Base_Conditionals)) As Integer

    Dim Input_Set(Size_of_Set) As String
    Dim Uncertainty_Sum As Double
    Dim Z_Order(Size_of_Set, Size_of_Set) As String
    Dim Test_Conditional As String
    Dim Test3 As Integer
    Dim Rank_Order(Size_of_Set, Size_of_Set) As String

    Dim Number_of_Z_Judgments As Integer = 0
    Dim Number_of_Zs_Judgments As Integer = 0
    Dim Number_of_P_Judgments As Integer = 0
    Dim Number_of_Classical_Judgments As Integer = 0
    Dim Number_of_Reflexive_Judgments As Integer = 0
    Dim Number_of_O_Judgments As Integer = 0
    Dim Number_of_O_Judgments_Holder As Integer = 0
    Dim Another_Number_of_O_Judgments_Holder As Integer = 0

    Dim Z_Score As Double = 0
    Dim Zs_Score As Double = 0
    Dim P_Score As Double = 0
    Dim Classical_Score As Double = 0
    Dim O_Score As Double = 0

    Dim Mean_Z_Score As Double
    Dim Mean_Zs_Score As Double
    Dim Mean_P_Score As Double
    Dim Mean_Classical_Score As Double
    Dim Mean_O_Score As Double

    Dim Punishment_Constant As Double = 0
    Dim PIR_Z_Score As Double = 0
    Dim PIR_Zs_Score As Double = 0
    Dim PIR_P_Score As Double = 0
    Dim PIR_Classical_Score As Double = 0
    Dim PIR_O_Score As Double = 0

    Dim SPunishment_Constant As Double = 0
    Dim Mean_PIR_Z_Score As Double
    Dim Mean_PIR_Zs_Score As Double
    Dim Mean_PIR_P_Score As Double
    Dim Mean_PIR_Classical_Score As Double
    Dim Mean_PIR_O_Score As Double

    Dim SPIR_Z_Score As Double = 0
    Dim SPIR_Zs_Score As Double = 0
    Dim SPIR_P_Score As Double = 0
```

```
Dim SPIR_Classical_Score As Double = 0
Dim SPIR_O_Score As Double = 0

Dim Mean_SPIR_Z_Score As Double
Dim Mean_SPIR_Zs_Score As Double
Dim Mean_SPIR_P_Score As Double
Dim Mean_SPIR_Classical_Score As Double
Dim Mean_SPIR_O_Score As Double

Dim Z_Errors As Integer = 0
Dim Zs_Errors As Integer = 0
Dim P_Errors As Integer = 0
Dim Classical_Errors As Integer = 0
Dim O_Errors As Integer = 0

Dim Candidate_Conclusion As String = ""
Dim Inference_Made_Test As Integer = 0

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim Test4 As Integer = 0
    Dim Test5 As Integer
    Dim Normicity_Counter As Integer = 0
    Dim Abs As Double

    Dim z_Proxy As Integer

    Proposition(1) = "a"
    Proposition(2) = "b|a"
    Proposition(3) = "b|~a"
    Proposition(4) = "c|a+b"
    Proposition(5) = "c|a+~b"
    Proposition(6) = "c|~a+b"
    Proposition(7) = "c|~a+~b"
    Proposition(8) = "d|a+b+c"
    Proposition(9) = "d|a+b+~c"
    Proposition(10) = "d|a+~b+c"
    Proposition(11) = "d|a+~b+~c"
    Proposition(12) = "d|~a+b+c"
    Proposition(13) = "d|~a+b+~c"
    Proposition(14) = "d|~a+~b+c"
    Proposition(15) = "d|~a+~b+~c"

    For v = 1 To Number_of_Runs

        For x = 1 To Number_of_Base_Conditionals + 1 'loads values into array to prevent unhandled exception.
            For y = 1 To 2 ^ (Number_of_Base_Conditionals + 1)
                P_Library(x, y) = "*"
            Next
        Next

        Test5 = 0
        While Test5 = 0

            Test5 = 1

            Test4 = 0
            While Test4 = 0
                Normicity_Counter = 0
                For x = 1 To 15
                    Probability(x) = get_value()
                    If Probability(x) >= 0.9 Then Normicity_Counter = Normicity_Counter + 1
                Next
                If Normicity_Counter >= Degree_of_Normicity Then
                    Test4 = 1
                    'load_P_Values() '!!!!! this and following line give an example of how to generate a probability distribution meeting certain constraints !!!!!!!
                    'If return_probability("(~b+c)-a)") <= 0.9 Then Test4 = 0
                End If
            End While

            ''!!!!!!!!!!!!!!!! this and the following lines may be used to fix a specific probability distribution !!!!!!!!!!!!!!!!!!!!!!!!!!!
            'To employ a given scenario remove the relevant “'”s to make code that is currenlty ‘commented out’ active

            If Scenario = 1 Then
                Proposition(2) = "b|a"
                Probability(2) = 0.05
                Proposition(3) = "b|~a"
                Probability(3) = 0.95
                Proposition(4) = "c|a+b"
                Probability(4) = 0.95
                Proposition(6) = "c|~a+b"
                Probability(6) = 0.05
            End If

            If Scenario = 2 Then
                Proposition(2) = "b|a"
                Probability(2) = 0.05
                Proposition(3) = "b|~a"
                Probability(3) = 0.95
                Proposition(4) = "c|a+b"
                Probability(4) = 0.95
                Proposition(6) = "c|~a+b"
                Probability(6) = 0.05
                Proposition(8) = "d|a+b+c"
                Probability(8) = 0.001
                Proposition(9) = "d|a+b+~c"
                Probability(9) = 0.999
            End If

            load_P_Values() '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

            'The following sub is used to query all 464 condititionals
            '**********************************************************************************************************************************************
            gererate_all_sensible_conds()
            '**********************************************************************************************************************************************

            'The following code is to set user inputed base conditionals
            '**********************************************************************************************************************************************

            If Scenario = 1 Then
                Clue_PPropositions(1) = "~c|b"
                Clue_Propositions(1) = trans_PProposition_to_Proposition(Clue_PPropositions(1))
                Clue_Probabilities(1) = return_probability(Clue_Propositions(1))

                Clue_PPropositions(2) = "c|a+b"
                Clue_Propositions(2) = trans_PProposition_to_Proposition(Clue_PPropositions(2))
                Clue_Probabilities(2) = return_probability(Clue_Propositions(2))
            End If

            If Scenario = 2 Then
                Clue_PPropositions(1) = "~c|b"
                Clue_Propositions(1) = trans_PProposition_to_Proposition(Clue_PPropositions(1))
                Clue_Probabilities(1) = return_probability(Clue_Propositions(1))

                Clue_PPropositions(2) = "c|a+b"
                Clue_Propositions(2) = trans_PProposition_to_Proposition(Clue_PPropositions(2))
                Clue_Probabilities(2) = return_probability(Clue_Propositions(2))
```

```
            Clue_PPropositions(3) = "~c|a+b+d"
            Clue_Propositions(3) = trans_PProposition_to_Proposition(Clue_PPropositions(3))
            Clue_Probabilities(3) = return_probability(Clue_Propositions(3))
        End If

        If Number_of_User_Inputed_Base_Conditionals > 0 Then
            For x = 1 To Number_of_User_Inputed_Base_Conditionals
                If Clue_Probabilities(x) < Min_Prob_for_Base_Conds Then Test5 = 0
            Next
        End If

        '*************************************************************************************************************************************************************

        'The following bit is used if we want to randomly generate premises.
        '*************************************************************************************************************************************************************
        Dim Counter As Integer
        Dim testy As Integer
        Counter = 0
        For x = 1 To 464  'Loop determines whether there is a sufficient number of high prob conds to serve as staring premises.
            If Queried_Probabilities(x) >= Min_Prob_for_Base_Conds Then Counter = Counter + 1
        Next
        If Counter < Number_of_Base_Conditionals Then Test5 = 0

        If Test5 = 1 Then
            For y = Number_of_User_Inputed_Base_Conditionals + 1 To Number_of_Base_Conditionals  '******* This loop randomly generates starting premises.
                testy = 1
                While testy = 1
                    testy = 0
                    gererate_datum()
                    If Datum_Prob >= Min_Prob_for_Base_Conds Then
                        For z = 1 To y - 1
                            If Clue_PPropositions(z) = Datum Then testy = 1
                        Next
                        If testy = 0 Then
                            Clue_PPropositions(y) = Datum
                            Clue_Probabilities(y) = Datum_Prob
                            Clue_Propositions(y) = trans_PProposition_to_Proposition(Clue_PPropositions(y))
                        End If
                    Else
                        testy = 1
                    End If
                End While
            Next
        End If
        '*************************************************************************************************************************************************************

        Size_of_Set = Number_of_Base_Conditionals
        Power_Set1 = powerset(Clue_Propositions)
        Power_Set1 = crunch_powerset(Power_Set1)
        Power_Set2 = powerset2(Clue_Probabilities)
        Power_Set2 = crunch_powerset2(Power_Set2)

    End While

    For y = 1 To Number_of_Props_Queried
        Z_LBounds(y) = 0
        Zs_LBounds(y) = 0
        P_LBounds(y) = 0
        Classical_LBounds(y) = 0
        Z_Judgments(y) = 0
        Zs_Judgments(y) = 0
        P_Judgments(y) = 0
        Classical_Judgments(y) = 0

        Test_Conditional = Queried_Propositions(y)

        If Test_Conditional = "c-~b" Then
            Test_Conditional = Test_Conditional
        End If

        If y = 1 Then
            For z = 1 To Number_of_Base_Conditionals + 1 'loads values into aray to prevent unhandled exception, and clears after second cycle.
                For w = 1 To 2 * (2 ^ (Number_of_Base_Conditionals + 1))
                    P_Library(z, w) = "*"
                Next
            Next
            P_Library_Count = 1
        Else
            For z = 1 To Number_of_Base_Conditionals + 1 'loads values into aray to prevent unhandled exception, and clears after second cycle.
                For w = 1 + (2 ^ (Number_of_Base_Conditionals + 1)) To 2 * (2 ^ (Number_of_Base_Conditionals + 1))
                    P_Library(z, w) = "*"
                Next
            Next
            P_Library_Count = 1 + (2 ^ (Number_of_Base_Conditionals + 1))
        End If

        For z = 1 To (2 ^ Number_of_Base_Conditionals) - 1

            z_Proxy = (2 ^ Number_of_Base_Conditionals) - z 'We use z_Proxy to count from value (2 ^ Number_of_Base_Conditionals) - 1 to 1.

            S_Size_of_Set = 1
            For a = 1 To Number_of_Base_Conditionals
                Input_Set(a) = Power_Set1(a, z_Proxy)
                If Not Input_Set(a) = "*" And Not Input_Set(a) = "" Then S_Size_of_Set = S_Size_of_Set + 1
            Next

            Uncertainty_Sum = 0
            For a = 1 To Number_of_Base_Conditionals
                If Not Input_Set(a) = "*" Then Uncertainty_Sum = Uncertainty_Sum + (1 - Power_Set2(a, z_Proxy))
            Next

            For r = 1 To Size_of_Set 'it is necessary to clear this set to prevent crashes.
                For s = 1 To Size_of_Set
                    Z_Order(r, s) = "*"
                    Rank_Order(r, s) = "*"
                Next
            Next


            If (1 - Uncertainty_Sum) > Z_LBounds(y) And Not (z = (2 ^ Number_of_Base_Conditionals) - 1) Then
                Test3 = 0

                system_Z_judgment()

                If Test3 = 1 Then
                    Z_Judgments(y) = 1
                    Z_LBounds(y) = 1 - Uncertainty_Sum
                    Zs_Judgments(y) = 1
                    Zs_LBounds(y) = 1 - Uncertainty_Sum
                End If
            End If

            If z = (2 ^ Number_of_Base_Conditionals) - 1 Then 'makes sure test Test_Conditional follows from full set of clues.
                Test3 = 0
                system_Z_judgment()
                If Test3 = 1 Then
                    If Not Z_Judgments(y) = 1 Then
```

```
                Z_Judgments(y) = 1
                Z_LBounds(y) = 1 - Uncertainty_Sum
                Zs_Judgments(y) = 1
                Zs_LBounds(y) = 1 - Uncertainty_Sum
            End If
        Else
            Z_Judgments(y) = 0
            Z_LBounds(y) = 0
        End If
    End If

    D_Size_of_Set = Size_of_Set
    Size_of_Set = S_Size_of_Set

    If (1 - Uncertainty_Sum) > P_LBounds(y) Then
        Test3 = 0

        system_P_judgment()

        If Test3 = 1 Then
            P_Judgments(y) = 1
            P_LBounds(y) = 1 - Uncertainty_Sum
        End If
    End If

    Size_of_Set = D_Size_of_Set

    If (1 - Uncertainty_Sum) > Classical_LBounds(y) Then
        Test3 = 0

        classical_judgment()

        If Test3 = 1 Then
            Classical_Judgments(y) = 1
            Classical_LBounds(y) = 1 - Uncertainty_Sum
        End If
    End If

Next

If Z_Judgments(y) = 1 Then
    If Z_LBounds(y) > Queried_Probabilities(y) Then
        Abs = (1 / 3) - (Z_LBounds(y) - Queried_Probabilities(y))
        Z_Errors = Z_Errors + 1
    Else
        Abs = (1 / 3) - (Queried_Probabilities(y) - Z_LBounds(y))
    End If
    Z_Score = Z_Score + Abs

    If Z_LBounds(y) > Queried_Probabilities(y) Then
        Abs = Punishment_Constant
    Else
        Abs = Z_LBounds(y)
    End If
    PIR_Z_Score = PIR_Z_Score + Abs

    If Z_LBounds(y) > Queried_Probabilities(y) Then
        Abs = (Queried_Probabilities(y) - Z_LBounds(y)) + SPunishment_Constant
    Else
        Abs = Z_LBounds(y)
    End If
    SPIR_Z_Score = SPIR_Z_Score + Abs

    Number_of_Z_Judgments = Number_of_Z_Judgments + 1
End If

If Zs_Judgments(y) = 1 Then
    If Zs_LBounds(y) > Queried_Probabilities(y) Then
        Abs = (1 / 3) - (Zs_LBounds(y) - Queried_Probabilities(y))
        Zs_Errors = Zs_Errors + 1
    Else
        Abs = (1 / 3) - (Queried_Probabilities(y) - Zs_LBounds(y))
    End If
    Zs_Score = Zs_Score + Abs

    If Zs_LBounds(y) > Queried_Probabilities(y) Then
        Abs = Punishment_Constant
    Else
        Abs = Zs_LBounds(y)
    End If
    PIR_Zs_Score = PIR_Zs_Score + Abs

    If Zs_LBounds(y) > Queried_Probabilities(y) Then
        Abs = (Queried_Probabilities(y) - Zs_LBounds(y)) + SPunishment_Constant
    Else
        Abs = Zs_LBounds(y)
    End If
    SPIR_Zs_Score = SPIR_Zs_Score + Abs

    Number_of_Zs_Judgments = Number_of_Zs_Judgments + 1
End If


If P_Judgments(y) = 1 Then
    If P_LBounds(y) > Queried_Probabilities(y) Then
        Abs = (1 / 3) - (P_LBounds(y) - Queried_Probabilities(y))
        P_Errors = P_Errors + 1
    Else
        Abs = (1 / 3) - (Queried_Probabilities(y) - P_LBounds(y))
    End If
    P_Score = P_Score + Abs

    If P_LBounds(y) > Queried_Probabilities(y) Then
        Abs = Punishment_Constant
    Else
        Abs = P_LBounds(y)
    End If
    PIR_P_Score = PIR_P_Score + Abs

    If P_LBounds(y) > Queried_Probabilities(y) Then
        Abs = (Queried_Probabilities(y) - P_LBounds(y)) + SPunishment_Constant
    Else
        Abs = P_LBounds(y)
    End If
    SPIR_P_Score = SPIR_P_Score + Abs

    Number_of_P_Judgments = Number_of_P_Judgments + 1
End If

If Classical_Judgments(y) = 1 Then
    If Classical_LBounds(y) > Queried_Probabilities(y) Then
        Abs = (1 / 3) - (Classical_LBounds(y) - Queried_Probabilities(y))
        Classical_Errors = Classical_Errors + 1
    Else
        Abs = (1 / 3) - (Queried_Probabilities(y) - Classical_LBounds(y))
    End If
    Classical_Score = Classical_Score + Abs
```

```
                If Classical_LBounds(y) > Queried_Probabilities(y) Then
                    Abs = Punishment_Constant
                Else
                    Abs = Classical_LBounds(y)
                End If
                PIR_Classical_Score = PIR_Classical_Score + Abs

                If Classical_LBounds(y) > Queried_Probabilities(y) Then
                    Abs = (Queried_Probabilities(y) - Classical_LBounds(y)) + SPunishment_Constant
                Else
                    Abs = Classical_LBounds(y)
                End If
                SPIR_Classical_Score = SPIR_Classical_Score + Abs

                Number_of_Classical_Judgments = Number_of_Classical_Judgments + 1
            End If
        Next

        'The following bit of code handles System O
        Inference_Made_Test = 1
        Number_of_O_Judgments = Number_of_Base_Conditionals

        For y = 1 To Number_of_Base_Conditionals
            O_Conclusions(y) = Clue_Propositions(y)
            O_LBounds(y) = Clue_Probabilities(y)
        Next

        While Inference_Made_Test = 1
            Number_of_O_Judgments_Holder = Number_of_O_Judgments
            Inference_Made_Test = 0
            right_weakening()
            very_cautious_monotony()
            exclusive_Or()
            wand()
        End While

        If v = 1 Then Another_Number_of_O_Judgments_Holder = 0
        For y = 1 To Number_of_Props_Queried
            For z = 1 To Number_of_O_Judgments
                If O_Conclusions(z) = Queried_Propositions(y) Then

                    Another_Number_of_O_Judgments_Holder = Another_Number_of_O_Judgments_Holder + 1

                    If O_LBounds(z) > Queried_Probabilities(y) Then
                        Abs = (1 / 3) - (O_LBounds(z) - Queried_Probabilities(y))
                        O_Errors = O_Errors + 1
                    Else
                        Abs = (1 / 3) - (Queried_Probabilities(y) - O_LBounds(z))
                    End If
                    O_Score = O_Score + Abs

                    If O_LBounds(z) > Queried_Probabilities(y) Then
                        Abs = Punishment_Constant
                    Else
                        Abs = O_LBounds(z)
                    End If
                    PIR_O_Score = PIR_O_Score + Abs

                    If O_LBounds(z) > Queried_Probabilities(y) Then
                        Abs = (Queried_Probabilities(y) - O_LBounds(z)) + SPunishment_Constant
                    Else
                        Abs = O_LBounds(z)
                    End If
                    SPIR_O_Score = SPIR_O_Score + Abs

                End If
            Next
        Next

    Next
    Number_of_O_Judgments = Another_Number_of_O_Judgments_Holder

    Mean_Z_Score = Z_Score / Number_of_Z_Judgments
    Mean_P_Score = P_Score / Number_of_P_Judgments
    Mean_Classical_Score = Classical_Score / Number_of_Classical_Judgments
    Mean_O_Score = O_Score / Number_of_O_Judgments

    Mean_PIR_Z_Score = PIR_Z_Score / Number_of_Z_Judgments
    Mean_PIR_P_Score = PIR_P_Score / Number_of_P_Judgments
    Mean_PIR_Classical_Score = PIR_Classical_Score / Number_of_Classical_Judgments
    Mean_PIR_O_Score = PIR_O_Score / Number_of_O_Judgments

    Mean_SPIR_Z_Score = SPIR_Z_Score / Number_of_Z_Judgments
    Mean_SPIR_P_Score = SPIR_P_Score / Number_of_P_Judgments
    Mean_SPIR_Classical_Score = SPIR_Classical_Score / Number_of_Classical_Judgments
    Mean_SPIR_O_Score = SPIR_O_Score / Number_of_O_Judgments

    Z_Errors = Z_Errors
    Zs_Errors = Zs_Errors
    P_Errors = P_Errors
    Classical_Errors = Classical_Errors
    O_Errors = O_Errors

    'the following is used to print results from a single run to a data file
    If Number_of_Runs = 1 Then
        load_consequence_sets()
        print_stuff()
    End If

    print_stuff2()

    If Number_of_O_Judgments > 3 And Number_of_P_Judgments > Number_of_O_Judgments Then
        Number_of_O_Judgments = Number_of_O_Judgments
    End If

End Sub

Private Sub system_P_judgment()
    Dim Test1 As Integer
    Dim Test2 As Integer = 2
    Dim Test4 As Integer
    Dim Step_Count As Integer
    Dim Row_Holder1(Size_of_Set) As String
    Dim Row_Holder2(Size_of_Set) As String
    Dim Row_Holder3 As String
    Dim Counter1 As Integer

    Row_Holder1(1) = Test_Conditional.Replace("-", "-~")
    For x = 2 To Size_of_Set
        Row_Holder1(x) = Input_Set(x - 1)
    Next

    Step_Count = 0
    Test1 = 0
    While Test1 = 0 'Loop determines size of Row_Holder1
        Step_Count = Step_Count + 1
```

```
        If Step_Count = Size_of_Set Then Test1 = 1
        If Row_Holder1(Step_Count) = "*" Then
            Test1 = 1
            Step_Count = Step_Count - 1
        End If
    End While

    If Test2 = 2 Then
        Test1 = 0
        While Test1 = 0
            Counter1 = 0
            Test4 = 0
            While Test4 = 0
                Counter1 = Counter1 + 1

                For w = 1 To Step_Count 'Loop makes duplicate of Row_Holder1
                    Row_Holder2(w) = Row_Holder1(w)
                Next

                Row_Holder2(Counter1) = Row_Holder2(Counter1).Replace("-", "+")  'Replaces "p-conds"s with "and"s, within element of row (powerset) in the que.
                For z = 1 To Step_Count
                    Row_Holder2(z) = Row_Holder2(z).Replace("-", ">") 'Replaces remaining "pcond"s with "mconds"s.
                Next
                Row_Holder3 = convert_to_PLstring(Row_Holder2, Step_Count)
                Row_Holder3 = remove_conditionals(Row_Holder3)
                Row_Holder3 = apply_deMorgan(Row_Holder3)
                Row_Holder3 = remove_double_negations(Row_Holder3)
                Row_Holder3 = associate(Row_Holder3)
                Row_Holder3 = simplify_CNF1(Row_Holder3)
                If Not resolve(Row_Holder3) Then
                    If Counter1 = 1 Then
                        Test2 = 0
                        Test4 = 1
                        Test1 = 1
                    Else
                        Test4 = 1
                        For x = Counter1 To Step_Count - 1
                            Row_Holder1(x) = Row_Holder1(x + 1)
                        Next
                        Row_Holder1(Step_Count) = "*"
                        Step_Count = Step_Count - 1
                    End If
                Else
                    If Counter1 = Step_Count And Test2 = 2 Then
                        Test2 = 1
                        Test4 = 1
                        Test1 = 1
                    End If
                End If
            End While

        End While

    End If

    If Test2 = 1 Then Test3 = 1
    Test2 = 2

End Sub

Private Sub classical_judgment()
    Dim Array_String(Size_of_Set + 1) As String
    Dim Test_String As String
    Dim Sub_Counter As Integer = 1

    Array_String(1) = "~(" & Test_Conditional & ")"
    Array_String(1) = Array_String(1).Replace("-", ">")
    For x = 2 To Size_of_Set + 1
        Array_String(x) = Input_Set(x - 1).Replace("-", ">")
        If Not Input_Set(x - 1) = "*" Then Sub_Counter = Sub_Counter + 1
    Next

    Test_String = convert_to_PLstring(Array_String, Sub_Counter)
    Test_String = remove_conditionals(Test_String)
    Test_String = apply_deMorgan(Test_String)
    Test_String = remove_double_negations(Test_String)
    Test_String = associate(Test_String)
    Test_String = simplify_CNF1(Test_String)
    If resolve(Test_String) Then Test3 = 1

End Sub

Private Sub system_Z_judgment()
    Dim Test1 As Integer
    Dim Test2 As Integer
    Dim Counter2 As Integer
    Dim Counter3 As Integer
    Dim Set_Holder_as_Formula As String
    Dim Set_Holder(Size_of_Set + 1) As String
    Generate_Alternate_Z_Ordering()
    Counter2 = 1
    Test3 = -1
    While Counter2 <= Size_of_Set And Test3 < 0
        Test1 = 0
        Test2 = 0
        Set_Holder(1) = Test_Conditional.Replace("-", "+")
        Counter3 = 0
        For x = 1 To Size_of_Set
            If Not Rank_Order(x, Counter2) = "" And Not Rank_Order(x, Counter2) = "*" Then
                Set_Holder(x + 1) = Rank_Order(x, Counter2).Replace("-", ">")
                Counter3 = Counter3 + 1
            End If
        Next
        Set_Holder_as_Formula = convert_to_PLstring(Set_Holder, Counter3 + 1)
        Set_Holder_as_Formula = remove_conditionals(Set_Holder_as_Formula)
        Set_Holder_as_Formula = apply_deMorgan(Set_Holder_as_Formula)
        Set_Holder_as_Formula = remove_double_negations(Set_Holder_as_Formula)
        Set_Holder_as_Formula = associate(Set_Holder_as_Formula)
        Set_Holder_as_Formula = simplify_CNF1(Set_Holder_as_Formula)
        If resolve(Set_Holder_as_Formula) = 0 Then ' resolve = 0 means set is consistent
            Test1 = 1
        End If
        Set_Holder(1) = Test_Conditional.Replace("-", "+~")
        Set_Holder_as_Formula = convert_to_PLstring(Set_Holder, Counter3 + 1)
        Set_Holder_as_Formula = remove_conditionals(Set_Holder_as_Formula)
        Set_Holder_as_Formula = apply_deMorgan(Set_Holder_as_Formula)
        Set_Holder_as_Formula = remove_double_negations(Set_Holder_as_Formula)
        Set_Holder_as_Formula = associate(Set_Holder_as_Formula)
        Set_Holder_as_Formula = simplify_CNF1(Set_Holder_as_Formula)
        If resolve(Set_Holder_as_Formula) = 0 Then
            Test2 = 1
        End If
        If Test1 = 1 And Test2 = 0 Then Test3 = 1
        Counter2 = Counter2 + 1
    End While

End Sub
```

```vb
Public Sub Generate_Alternate_Z_Ordering()
    Dim Column_Spot As Integer
    Dim Counter1 As Integer
    Dim Test9 As Integer
    Dim Rank_Spot As Integer
    Dim Membership_Count As Integer
    Dim Z_Rank As Integer
    Dim Set_Holder_as_Formula As String
    Dim Set_Holder(Size_of_Set) As String
    Dim Revised_Input_Set(Size_of_Set) As String
    Dim Input_Set_Copy(Size_of_Set) As String
    Dim Orig_Size_of_Set As Integer = Size_of_Set
    Dim Set_is_EpsInconsistent As Integer = 0
    Z_Rank = 1
    Rank_Spot = 1
    Counter1 = 0
    Test9 = 0
    While Test9 = 0
        Counter1 = Counter1 + 1
        If Not Input_Set(Counter1) = "" And Not Input_Set(Counter1) = "*" Then
            Revised_Input_Set(Counter1) = Input_Set(Counter1)
            Input_Set_Copy(Counter1) = Input_Set(Counter1)
        Else
            Test9 = 1
            Counter1 = Counter1 - 1
        End If
        If Counter1 = Size_of_Set Then
            Test9 = 1
            'Counter1 = Counter1 - 1
        End If
    End While

    Size_of_Set = Counter1

    While Size_of_Set > 0 'This 'while loop' generates the Z-ordering for the input set.  Conditionals along the y-axis (rows) or of the same rank.

        For i = 1 To Size_of_Set
            For w = 1 To Size_of_Set
                Set_Holder(w) = Input_Set_Copy(w)
            Next
            Set_Holder(i) = Set_Holder(i).Replace("-", "+")
            For z = 1 To Size_of_Set
                Set_Holder(z) = Set_Holder(z).Replace("-", ">")
            Next
            Set_Holder_as_Formula = convert_to_PLstring(Set_Holder, Size_of_Set)
            Set_Holder_as_Formula = remove_conditionals(Set_Holder_as_Formula)
            Set_Holder_as_Formula = apply_deMorgan(Set_Holder_as_Formula)
            Set_Holder_as_Formula = remove_double_negations(Set_Holder_as_Formula)
            Set_Holder_as_Formula = associate(Set_Holder_as_Formula)
            Set_Holder_as_Formula = simplify_CNF1(Set_Holder_as_Formula)
            If Not resolve(Set_Holder_as_Formula) Then  '"Not resolve" means consistent.
                Z_Order(Rank_Spot, Z_Rank) = Input_Set_Copy(i)
                Rank_Spot = Rank_Spot + 1
                Revised_Input_Set(i) = "*"
                Membership_Count = Membership_Count + 1
            End If
        Next

        If Membership_Count = 0 Then
            Set_is_EpsInconsistent = 1
            For nl = 1 To Size_of_Set
                Z_Order(nl, Z_Rank) = Input_Set_Copy(nl)
            Next
            Size_of_Set = 0
        End If

        If Set_is_EpsInconsistent = 0 Then
            Z_Rank = Z_Rank + 1
            Rank_Spot = 1
            Counter1 = 1
            For j = 1 To Size_of_Set
                If Not (Revised_Input_Set(j) = "*") Then
                    Input_Set_Copy(Counter1) = Revised_Input_Set(j) '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                    Counter1 = Counter1 + 1
                End If
            Next
            Size_of_Set = Size_of_Set - Membership_Count
            Membership_Count = 0
            For w = 1 To Size_of_Set
                Revised_Input_Set(w) = Input_Set_Copy(w)
            Next
        End If

    End While


    Size_of_Set = Orig_Size_of_Set
    Z_Rank = Z_Rank - 1
    Column_Spot = 0 'This part of the code reorders the Z_Ordering.
    For x = 1 To Z_Rank 'loops through the rows of the Z-ordering
        Membership_Count = 0
        Test9 = 1
        Counter1 = 1
        While Test9 = 1 'counts number of elements of Z_Order in row x.
            If Not Z_Order(Counter1, (Z_Rank + 1) - x) = "" And Not Z_Order(Counter1, (Z_Rank + 1) - x) = "*" Then
                Membership_Count = Membership_Count + 1
            Else
                Test9 = 0
            End If
            If Counter1 = Size_of_Set Then Test9 = 0
            Counter1 = Counter1 + 1
        End While
        For y = x To Z_Rank
            Test9 = 1
            Counter1 = 1
            While Test9 = 1
                If Counter1 <= Membership_Count And Not Membership_Count = 0 Then
                    Rank_Order(Column_Spot + Counter1, y) = Z_Order(Counter1, (Z_Rank + 1) - x)
                Else
                    Test9 = 0
                End If
                Counter1 = Counter1 + 1
            End While
        Next
        Column_Spot = Column_Spot + Membership_Count
    Next

End Sub

Public Function resolve(ByVal Val As String) As Boolean
    Dim Output As Boolean = False
    Dim Step_Count As Integer = 1
    Dim Clause_Count1 As Integer = 1
    Dim Clause_Count2 As Integer
    Dim Clause_Count3 As Integer
    Dim Literal_Count1 As Integer = 1
```

```
Dim Literal_Count2 As Integer
Dim Max_Clause_Length As Integer = 25 '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Dim Form_Size As Integer = 200 '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Dim Form(Form_Size, Max_Clause_Length) As String
Dim Place_Counter(Form_Size) As Integer
Dim Test1 As Integer = 0
Dim Test2 As Integer = 0
Dim Test3 As Integer = 0
Dim Holder As Integer
Dim Clause_Holder(Max_Clause_Length) As String
Dim Testx As Integer
Dim Counterx As Integer

For x = 1 To Form_Size
    Place_Counter(x) = x + 1  'Will record up to what clause the corresponsing cell of the array has been tested against.
Next

For x = 1 To Form_Size  'Fills for with "**", so all cells can be accessed by string tests
    For y = 1 To Max_Clause_Length
        Form(x, y) = "**"
    Next
Next

While Step_Count < Val.Length 'This loop loads a matrix which records the clauses and literals of the formula for which resolution will be attempted.
    If Not (UCase(Val.Chars(Step_Count)) = LCase(Val.Chars(Step_Count))) And Val.Chars(Step_Count - 1) = "~" Then
        Form(Clause_Count1, Literal_Count1) = Val.Chars(Step_Count) & "~"
        Literal_Count1 = Literal_Count1 + 1
    End If
    If Not (UCase(Val.Chars(Step_Count)) = LCase(Val.Chars(Step_Count))) And Not (Val.Chars(Step_Count - 1) = "~") Then
        Form(Clause_Count1, Literal_Count1) = Val.Chars(Step_Count) & "*"
        Literal_Count1 = Literal_Count1 + 1
    End If
    If Val.Chars(Step_Count) = "+" Then
        Clause_Count1 = Clause_Count1 + 1
        Literal_Count1 = 1
    End If
    Step_Count = Step_Count + 1
End While

Clause_Count3 = Clause_Count1 'Clause_Count3 will record the number of clauses that have been 'accepted' in Form at any given point.

Counterx = 0
For x = 1 To Clause_Count3 'This loop eliminates redundancies in the initial clauses.
    For y = 1 To Max_Clause_Length
        Clause_Holder(y) = Form(x, y)
    Next
    Clause_Holder = simplify4(Clause_Holder, Max_Clause_Length, Clause_Count3, Form, x)
    For y = 1 To Max_Clause_Length
        Form(x, y) = "**"
    Next
    If Clause_Holder(1) = "**" Then
        Counterx = Counterx + 1
    Else
        For y = 1 To Max_Clause_Length
            Form(x - Counterx, y) = Clause_Holder(y)
        Next
    End If
Next

Clause_Count3 = Clause_Count3 - Counterx 'Reset Clause_Count3.

Clause_Count1 = 1
Literal_Count1 = 1
Clause_Count2 = 2
Literal_Count2 = 1

Test1 = 0
While Test1 = 0  ' Main Cycle: Clause_Count1, Literal_Count1 and Clause_Count2, Literal_Count2 'point' to the elements of clauses that are checked for matching.
    Test1 = 1
    Test2 = 0

    Counterx = 1
    Testx = 0
    While Testx = 0
        If Place_Counter(Counterx) < Clause_Count3 Then
            Clause_Count1 = Counterx
            Literal_Count1 = 1
            Clause_Count2 = Place_Counter(Clause_Count1)
            Literal_Count2 = 1
            Testx = 1
        End If
        If Counterx = Clause_Count3 Then Testx = 1
        Counterx = Counterx + 1
    End While

    While Test2 = 0
        If Literal_Count2 > 1 And Form(Clause_Count2, Literal_Count2) = "**" Then
            Clause_Count2 = Clause_Count2 + 1
            Literal_Count2 = 1
            If Form(Clause_Count1, Literal_Count1 + 1) = "**" Then
                Place_Counter(Clause_Count1) = Clause_Count2 - 1
                Test1 = Test1
            End If
        End If
        If Form(Clause_Count2, 1) = "**" Then
            Literal_Count1 = Literal_Count1 + 1
            Clause_Count2 = Place_Counter(Clause_Count1)
            Literal_Count2 = 1
        End If
        If Literal_Count1 > 1 And Form(Clause_Count1, Literal_Count1) = "**" Then
            Clause_Count1 = Clause_Count1 + 1
            Clause_Count2 = Place_Counter(Clause_Count1)
            Literal_Count1 = 1
            Literal_Count2 = 1
        End If
        If Form(Clause_Count1, 1) = "**" Then
            Test2 = 1
        End If

        If Clause_Count1 = 1 And Literal_Count1 = 1 And Clause_Count2 = 3 And Literal_Count2 = 2 Then ' conditional for testing purposes
            Clause_Count1 = Clause_Count1
        End If

        If Form(Clause_Count1, Literal_Count1).Chars(0) = Form(Clause_Count2, Literal_Count2).Chars(0) And _
        Not (Form(Clause_Count1, Literal_Count1).Chars(1) = Form(Clause_Count2, Literal_Count2).Chars(1)) Then
            If Form(Clause_Count1, 2) = "**" And Form(Clause_Count2, 2) = "**" Then
                Test1 = 1
                Test2 = 1
                Output = True
            Else
                Step_Count = 1
                Test3 = 0
                While Test3 = 0
                    If Step_Count < Literal_Count1 Then
                        Form(Clause_Count3 + 1, Step_Count) = Form(Clause_Count1, Step_Count)
                    End If
                    If Step_Count > Literal_Count1 Then
```

```
                            Form(Clause_Count3 + 1, Step_Count - 1) = Form(Clause_Count1, Step_Count)
                        End If
                        Step_Count = Step_Count + 1
                        If Form(Clause_Count1, Step_Count) = "**" Then
                            Test3 = 1
                        End If
                    End While
                    Holder = Step_Count - 1
                    Test3 = 0
                    Step_Count = 1
                    While Test3 = 0
                        If Step_Count < Literal_Count2 Then
                            Form(Clause_Count3 + 1, Step_Count + (Holder - 1)) = Form(Clause_Count2, Step_Count)
                        End If
                        If Step_Count > Literal_Count2 Then
                            Form(Clause_Count3 + 1, Step_Count + (Holder - 2)) = Form(Clause_Count2, Step_Count)
                        End If
                        Step_Count = Step_Count + 1
                        If Form(Clause_Count2, Step_Count) = "**" Then
                            Test3 = 1
                        End If
                    End While
                    For x = 1 To Max_Clause_Length
                        Clause_Holder(x) = Form(Clause_Count3 + 1, x)
                    Next
                    Clause_Holder = simplify1(Clause_Holder, Max_Clause_Length)
                    Clause_Holder = simplify2(Clause_Holder, Max_Clause_Length)
                    Clause_Holder = simplify3(Clause_Holder, Max_Clause_Length, Clause_Count3, Form)
                    For x = 1 To Max_Clause_Length
                        Form(Clause_Count3 + 1, x) = Clause_Holder(x)
                    Next
                    If Not (Clause_Holder(1) = "**") Then Clause_Count3 = Clause_Count3 + 1
                    Clause_Count2 = Clause_Count2 + 1
                    Literal_Count2 = 0
                End If
            End If
            If Test2 = 0 Then Literal_Count2 = Literal_Count2 + 1
        End While
    End While
    Return Output
End Function

Public Function simplify1(ByVal Val As Array, ByVal Max_Clause_Length As Integer) As Array 'sub checks for redundant literals within a proposed addition to form
    Dim Output(Max_Clause_Length) As String
    Dim Test1 As Integer = 0
    Dim Step_Count As Integer = 1
    For x = 1 To Max_Clause_Length
        For y = x + 1 To Max_Clause_Length
            If Val(x) = Val(y) Then
                Val(x) = "*"
                y = y + 1
            End If
        Next
    Next
    For x = 1 To Max_Clause_Length
        If Not (Val(x) = "*") Then
            If Not (Val(x) = "**") Then
                Output(Step_Count) = Val(x)
                Step_Count = Step_Count + 1
            End If
        End If
    Next
    For x = Step_Count To Max_Clause_Length
        Output(x) = "**"
    Next

    Return Output
End Function

Public Function simplify2(ByVal Val As Array, ByVal Max_Clause_Length As Integer) As Array 'checks for resolution (following simplify1)
    Dim Test1 As Boolean = False
    Dim Step_Count As Integer = 1

    For x = 1 To Max_Clause_Length
        For y = x + 1 To Max_Clause_Length
            If Not (Val(x) = "**") And Val(x).Chars(0) = Val(y).Chars(0) Then Test1 = True
        Next
    Next

    If Test1 = True Then
        For x = 1 To Max_Clause_Length
            Val(x) = "**"
        Next
    End If

    Return Val
End Function

Public Function simplify3(ByVal Val As Array, ByVal Max_Clause_Length As Integer, ByVal Clause_Count3 As Integer, ByVal Form As Array) As Array 'sub checks whether row was
already added
    Dim Test1 As Integer = 0
    Dim Test2 As Integer
    Dim Step_Count As Integer = 0
    Dim Val_Length As Integer
    Dim X_Length As Integer
    Dim Countery As Integer

    While Test2 = 0
        Step_Count = Step_Count + 1
        If Val(Step_Count) = "**" Then
            Test2 = 1
        End If
    End While
    Val_Length = Step_Count
    For X = 1 To Clause_Count3
        Test2 = 0
        Step_Count = 0
        While Test2 = 0
            Step_Count = Step_Count + 1
            If Form(X, Step_Count) = "**" Then
                Test2 = 1
            End If
        End While
        X_Length = Step_Count
        If X_Length <= Val_Length Then
            Countery = 0
            For Y = 1 To Val_Length
                Test2 = 0
                For Z = 1 To X_Length
                    If Not (Val(Y) = "**") And Val(Y) = Form(X, Z) Then Countery = Countery + 1
                Next
                If Countery = X_Length - 1 Then Test1 = 1
            Next
        End If
    Next
    If Test1 = 1 Then
        For x = 1 To Max_Clause_Length
```

```
                Val(x) = "**"
            Next
        End If
        Return Val
    End Function

    Public Function simplify4(ByVal Val As Array, ByVal Max_Clause_Length As Integer, ByVal Clause_Count3 As Integer, ByVal Form As Array, ByVal Test_Clause_Position As Integer)
As Array
        Dim Test1 As Integer = 0
        Dim Test2 As Integer
        Dim Step_Count As Integer = 0
        Dim Val_Length As Integer
        Dim X_Length As Integer
        Dim Countery As Integer

        While Test2 = 0
            Step_Count = Step_Count + 1
            If Val(Step_Count) = "**" Then
                Test2 = 1
            End If
        End While

        Val_Length = Step_Count
        For X = 1 To Clause_Count3

            Test2 = 0
            Step_Count = 0
            While Test2 = 0
                Step_Count = Step_Count + 1
                If Form(X, Step_Count) = "**" Or X = Test_Clause_Position Then
                    Test2 = 1
                End If
            End While

            X_Length = Step_Count
            If X_Length <= Val_Length And Not X = Test_Clause_Position Then
                Countery = 0
                For Y = 1 To Val_Length
                    Test2 = 0
                    For Z = 1 To X_Length
                        If Not (Val(Y) = "**") And Val(Y) = Form(X, Z) Then Countery = Countery + 1
                    Next
                    If Countery = X_Length - 1 And Not Form(X, 1) = "**" Then Test1 = 1
                Next
            End If

        Next
        If Test1 = 1 Then
            For x = 1 To Max_Clause_Length
                Val(x) = "**"
            Next
        End If
        Return Val
    End Function

    Public Function crunch_powerset(ByVal Power_Set_Holder As Array) As Array
        Dim Step_Count As Integer = Size_of_Set
        Dim Test As Integer = 0
        While Test = 0
            Test = 1
            For y = 1 To 2 ^ Size_of_Set
                Step_Count = Size_of_Set
                While Step_Count > 0
                    If Not (Power_Set_Holder(Step_Count, y) = "*") Then
                        If Power_Set_Holder(Step_Count - 1, y) = "*" Then
                            Power_Set_Holder(Step_Count - 1, y) = Power_Set_Holder(Step_Count, y)
                            Power_Set_Holder(Step_Count, y) = "*"
                            Test = 0
                        End If
                    End If
                    Step_Count = Step_Count - 1
                End While
            Next
        End While
        Return Power_Set_Holder
    End Function

    Public Function powerset(ByVal Input_Set As Array) As Array
        Dim Power_Set_Holder(Size_of_Set, 2 ^ Size_of_Set) As String
        Dim Step_Count1 As Integer
        Dim Step_Count2 As Integer
        Dim Test_Value As Integer
        Dim Toggle As Integer
        For x = 1 To Size_of_Set
            For y = 1 To 2 ^ Size_of_Set
                Power_Set_Holder(x, y) = "*"
            Next
        Next
        For x = 1 To Size_of_Set
            Test_Value = 2 ^ (Size_of_Set - x)
            Toggle = 1
            Step_Count1 = 1
            Step_Count2 = 1
            While Step_Count1 <= 2 ^ Size_of_Set
                If Toggle = 1 Then Power_Set_Holder(x, Step_Count1) = Input_Set(x)
                If Step_Count2 = Test_Value And Toggle = 1 Then
                    Toggle = 0
                    Step_Count2 = 0
                End If
                If Step_Count2 = Test_Value And Toggle = 0 Then
                    Toggle = 1
                    Step_Count2 = 0
                End If
                Step_Count1 = Step_Count1 + 1
                Step_Count2 = Step_Count2 + 1
            End While
        Next
        Return Power_Set_Holder
    End Function

    Public Function crunch_powerset2(ByVal Power_Set_Holder As Array) As Array
        Dim Step_Count As Integer = Size_of_Set
        Dim Test As Integer = 0
        While Test = 0
            Test = 1
            For y = 1 To 2 ^ Size_of_Set
                Step_Count = Size_of_Set
                While Step_Count > 0
                    If Not (Power_Set_Holder(Step_Count, y) = -0.000001) Then
                        If Power_Set_Holder(Step_Count - 1, y) = -0.000001 Then
                            Power_Set_Holder(Step_Count - 1, y) = Power_Set_Holder(Step_Count, y)
                            Power_Set_Holder(Step_Count, y) = -0.000001
                            Test = 0
                        End If
                    End If
                    Step_Count = Step_Count - 1
                End While
```

```
        Next
    End While
    Return Power_Set_Holder
End Function

Public Function powerset2(ByVal Input_Set As Array) As Array
    Dim Power_Set_Holder(Size_of_Set, 2 ^ Size_of_Set) As Double
    Dim Step_Count1 As Integer
    Dim Step_Count2 As Integer
    Dim Test_Value As Integer
    Dim Toggle As Integer
    For x = 1 To Size_of_Set
        For y = 1 To 2 ^ Size_of_Set
            Power_Set_Holder(x, y) = -0.000001
        Next
    Next
    For x = 1 To Size_of_Set
        Test_Value = 2 ^ (Size_of_Set - x)
        Toggle = 1
        Step_Count1 = 1
        Step_Count2 = 1
        While Step_Count1 <= 2 ^ Size_of_Set
            If Toggle = 1 Then Power_Set_Holder(x, Step_Count1) = Input_Set(x)
            If Step_Count2 = Test_Value And Toggle = 1 Then
                Toggle = 0
                Step_Count2 = 0
            End If
            If Step_Count2 = Test_Value And Toggle = 0 Then
                Toggle = 1
                Step_Count2 = 0
            End If
            Step_Count1 = Step_Count1 + 1
            Step_Count2 = Step_Count2 + 1
        End While
    Next
    Return Power_Set_Holder
End Function

Public Function convert_to_PLstring(ByVal Row_Holder1 As Array, ByVal Test_Set_Length As Integer) As String
    Dim Output As String = ""
    If Test_Set_Length = 1 Then
        Output = "(" & Row_Holder1(1) & ")"
    End If
    If Test_Set_Length > 1 Then
        For X = 1 To Test_Set_Length - 1
            Output = Output & "("
        Next
        Output = Output & "(" & Row_Holder1(1) & ")"
        For X = 2 To Test_Set_Length
            Output = Output & "+(" & Row_Holder1(X) & "))"
        Next
    End If
    Output = Output.Substring(1, Output.Length - 2)
    Return Output
End Function

Public Function remove_conditionals(ByVal Val As String) As String
    Dim Reference_Position As Integer
    Dim Step_Counter As Integer
    Dim Val_Copy As String = Val
    Dim Test10 As Integer = 0
    Dim Test11 As Integer
    Dim Light_Brace_Counter As Integer
    Dim Right_Brace_Counter As Integer
    While Test10 = 0
        Reference_Position = 0
        Test11 = 0
        While Test11 = 0
            If Val.Chars(Reference_Position) = ">" Then
                Step_Counter = 0
                Right_Brace_Counter = 0
                Light_Brace_Counter = 0
                If Val.Chars(Reference_Position - 1) = ")" Then
                    While Test11 = 0
                        Step_Counter = Step_Counter + 1
                        If Val.Chars(Reference_Position - Step_Counter) = ")" Then Right_Brace_Counter = Right_Brace_Counter + 1
                        If Val.Chars(Reference_Position - Step_Counter) = "(" Then Light_Brace_Counter = Light_Brace_Counter + 1
                        If Right_Brace_Counter = Light_Brace_Counter Then Test11 = 1
                    End While
                    If Reference_Position - Step_Counter = 0 Then
                        Val_Copy = "~" & Val.Substring(0, Step_Counter) & "|" & Val.Substring(Reference_Position + 1, Val.Length - (Reference_Position + 1))
                    Else
                        Val_Copy = Val.Substring(0, Reference_Position - Step_Counter) & "~" & Val.Substring(Reference_Position - Step_Counter, Step_Counter) & _
                        "|" & Val.Substring(Reference_Position + 1, Val.Length - (Reference_Position + 1))
                    End If
                Else
                    While Test11 = 0 And Reference_Position - Step_Counter > 0
                        Step_Counter = Step_Counter + 1
                        If Val.Chars(Reference_Position - Step_Counter) = "(" Or Reference_Position - Step_Counter = 0 Then Test11 = 1
                    End While
                    If Reference_Position - Step_Counter = 0 And Not (Val.Chars(Reference_Position - Step_Counter) = "(") Then
                        Val_Copy = "~" & Val.Substring(0, Reference_Position - Step_Counter + 1) & "|" & _
                        Val.Substring(Reference_Position + 1, Val.Length - (Reference_Position + 1))
                    Else
                        Val_Copy = Val.Substring(0, Reference_Position - Step_Counter + 1) & "~" & _
                        Val.Substring(Reference_Position - Step_Counter + 1, Step_Counter - 1) & "|" & _
                        Val.Substring(Reference_Position + 1, Val.Length - (Reference_Position + 1))
                    End If
                End If
            End If
            Reference_Position = Reference_Position + 1
            If Reference_Position = Val.Length Then Test11 = 1
        End While
        If Val = Val_Copy Then Test10 = 1
        Val = Val_Copy
    End While
    Return Val
End Function

Public Function apply_deMorgan(ByVal Val As String) As String
    Dim Val_Copy As String = Val
    Dim Position1 As Integer
    Dim Position2 As Integer
    Dim Position3 As Integer
    Dim Connective As String
    Dim Test1 As Integer = 0
    Dim Test2 As Integer
    Dim Test3 As Integer
    Dim Reference_Position As Integer
    Dim Step_Counter As Integer
    Dim Light_Brace_Counter As Integer
    Dim Right_Brace_Counter As Integer
    While Test1 = 0
        Reference_Position = 0
        Test2 = 0
        Test3 = 0
        If Val_Copy.Length = 1 Then Test2 = 1
```

```
        While Test2 = 0
            If Val.Chars(Reference_Position) = "~" And Val.Chars(Reference_Position + 1) = "(" Then
                Position1 = Reference_Position + 1
                Step_Counter = 1
                Right_Brace_Counter = 0
                Light_Brace_Counter = 0
                While Test2 = 0
                    Step_Counter = Step_Counter + 1
                    If Val.Chars(Reference_Position + Step_Counter) = "(" Then Light_Brace_Counter = Light_Brace_Counter + 1
                    If Val.Chars(Reference_Position + Step_Counter) = ")" Then Right_Brace_Counter = Right_Brace_Counter + 1
                    If Light_Brace_Counter > 0 And Light_Brace_Counter = Right_Brace_Counter Then
                        Test2 = 1
                        Position2 = Reference_Position + Step_Counter + 1
                        Step_Counter = Step_Counter + 1
                    End If
                    If Light_Brace_Counter = 0 And Not (UCase(Val.Chars(Reference_Position + Step_Counter)) = LCase(Val.Chars(Reference_Position + Step_Counter))) Then
                        Test2 = 1
                        Position2 = Reference_Position + Step_Counter + 1
                        Step_Counter = Step_Counter + 1
                    End If
                End While
                While Test3 = 0
                    Step_Counter = Step_Counter + 1
                    If Val.Chars(Reference_Position + Step_Counter) = "(" Then Light_Brace_Counter = Light_Brace_Counter + 1
                    If Val.Chars(Reference_Position + Step_Counter) = ")" Then Right_Brace_Counter = Right_Brace_Counter + 1
                    If Light_Brace_Counter > 0 And Light_Brace_Counter = Right_Brace_Counter Then
                        Test3 = 1
                        Position3 = Reference_Position + Step_Counter + 1
                    End If
                    If Light_Brace_Counter = 0 And Not (UCase(Val.Chars(Reference_Position + Step_Counter)) = LCase(Val.Chars(Reference_Position + Step_Counter))) Then
                        Test3 = 1
                        Position3 = Reference_Position + Step_Counter + 1
                    End If
                End While
                If Val.Chars(Position2) = "|" Then
                    Connective = "+"
                Else
                    Connective = "|"
                End If
                If Position1 < 2 Then Val_Copy = "(~" & Val.Substring(Position1 + 1, Position2 - (Position1 + 1)) & Connective & "~" & _
                Val.Substring(Position2 + 1, Position3 - (Position2 + 1)) & Val.Substring(Position3, Val.Length - Position3)
                If Position1 > 1 Then Val_Copy = Val.Substring(0, Position1 - 1) & "(~" & Val.Substring(Position1 + 1, Position2 - (Position1 + 1)) & _
                Connective & "~" & Val.Substring(Position2 + 1, Position3 - (Position2 + 1)) & Val.Substring(Position3, Val.Length - Position3)
            End If
            Reference_Position = Reference_Position + 1
            If Reference_Position + 1 = Val.Length Then Test2 = 1
        End While
        If Val = Val_Copy Then Test1 = 1
        Val = Val_Copy
    End While
    Return Val
End Function

Public Function remove_double_negations(ByVal Val As String) As String
    Dim Reference_Position As Integer
    Dim Val_Copy As String = Val
    Dim Test1 As Integer = 0
    Dim Test2 As Integer
    While Test1 = 0
        Reference_Position = 0
        Test2 = 0
        If Val_Copy.Length = 1 Then Test2 = 1
        While Test2 = 0 And Val.Length > 1
            If Val.Chars(Reference_Position) = "~" And Val.Chars(Reference_Position + 1) = "~" And Not (Reference_Position = 0) Then
                Val_Copy = Val.Substring(0, Reference_Position) & Val.Substring(Reference_Position + 2, Val.Length - (Reference_Position + 2))
            End If
            If Val.Chars(Reference_Position) = "~" And Val.Chars(Reference_Position + 1) = "~" And Reference_Position = 0 Then
                Val_Copy = Val.Substring(Reference_Position + 2, Val.Length - (Reference_Position + 2))
            End If
            Reference_Position = Reference_Position + 1
            If Reference_Position = Val.Length - 1 Then Test2 = 1
        End While
        If Val = Val_Copy Then Test1 = 1
        Val = Val_Copy
    End While
    Return Val
End Function

Public Function associate(ByVal Val As String) As String
    Dim Val_Copy As String = Val
    Dim Test1 As Integer = 0
    Dim Test2 As Integer = 0
    Dim Left_Check(3) As Integer
    Dim Right_Check(3) As Integer
    Dim Reference_Position As Integer
    While Test1 = 0
        Reference_Position = 1
        Test2 = 0
        If Val_Copy.Length = 1 Then
            Test1 = 1
            Test2 = 1
        End If
        While Test2 = 0
            If Val.Chars(Reference_Position) = "|" And Val.Chars(Reference_Position - 1) = ")" Then
                Left_Check = left_subwff_plus(Val, Reference_Position - 1)
                If Left_Check(0) = 1 Then
                    Right_Check = right_subwff(Val, Reference_Position + 1)
                    Val_Copy = Val.Substring(0, Left_Check(1)) & "(" & Val.Substring(Left_Check(1) + 1, Left_Check(2) - (Left_Check(1) + 1)) & "|" & _
                    Val.Substring(Right_Check(1), Right_Check(3) - (Right_Check(1) - 1)) & ")+(" & Val.Substring(Left_Check(2) + 1, Left_Check(3) - (Left_Check(2) + 1)) _
                    & "|" & Val.Substring(Right_Check(1), Right_Check(3) - (Right_Check(1) - 1)) & ")" & Val.Substring(Right_Check(3) + 1, Val.Length - (Right_Check(3) + 1))
                    Test2 = 1
                End If
            End If
            If Test2 = 0 And Val.Chars(Reference_Position) = "|" And Val.Chars(Reference_Position + 1) = "(" Then
                Right_Check = right_subwff_plus(Val, Reference_Position + 1)
                If Right_Check(0) = 1 Then
                    Left_Check = left_subwff(Val, Reference_Position - 1)
                    Val_Copy = Val.Substring(0, Left_Check(1)) & "(" & Val.Substring(Left_Check(1), Left_Check(3) - (Left_Check(1) - 1)) & "|" & _
                    Val.Substring(Right_Check(1) + 1, Right_Check(2) - (Right_Check(1) + 1)) & ")+(" & _
                    Val.Substring(Left_Check(1), Left_Check(3) - (Left_Check(1) - 1)) & "|" & Val.Substring(Right_Check(2) + 1, Right_Check(3) - (Right_Check(2) + 1)) _
                    & ")" & Val.Substring(Right_Check(3) + 1, Val.Length - (Right_Check(3) + 1))
                    Test2 = 1
                End If
            End If
            If Reference_Position = Val.Length - 2 And Val = Val_Copy Then
                Test2 = 1
                Test1 = 1
            End If
            Reference_Position = Reference_Position + 1
            Val = Val_Copy
        End While
    End While
    Return Val
End Function

Public Function left_subwff_plus(ByVal Val As String, ByVal Reference_Position As Integer) As Array
    Dim Left_Brace_Counter As Integer = 0
```

```
    Dim Right_Brace_Counter As Integer = 0
    Dim Step_Count As Integer = 0
    Dim Test1 As Integer = 0
    Dim Holder(3) As Integer
    Holder(3) = Reference_Position
    While Test1 = 0
        If Val.Chars(Reference_Position - Step_Count) = ")" Then Right_Brace_Counter = Right_Brace_Counter + 1
        If Val.Chars(Reference_Position - Step_Count) = "(" Then Left_Brace_Counter = Left_Brace_Counter + 1
        If Left_Brace_Counter = Right_Brace_Counter Then
            Holder(1) = Reference_Position - Step_Count
            Test1 = 1
        End If
        Step_Count = Step_Count + 1
    End While
    Test1 = 0
    Step_Count = Holder(1)
    Left_Brace_Counter = 0
    Right_Brace_Counter = 0
    While Test1 = 0
        If Val.Chars(Step_Count) = "(" Then Left_Brace_Counter = Left_Brace_Counter + 1
        If Val.Chars(Step_Count) = ")" Then Right_Brace_Counter = Right_Brace_Counter + 1
        If Left_Brace_Counter = Right_Brace_Counter + 1 And (Val.Chars(Step_Count) = "+" Or Val.Chars(Step_Count) = "|") Then
            Holder(2) = Step_Count
            If Val.Chars(Step_Count) = "+" Then Holder(0) = 1
            Test1 = 1
        End If
        Step_Count = Step_Count + 1
    End While
    Return Holder
End Function

Public Function right_subwff_plus(ByVal Val As String, ByVal Reference_Position As Integer) As Array
    Dim Left_Brace_Counter As Integer = 0
    Dim Right_Brace_Counter As Integer = 0
    Dim Step_Count As Integer = 0
    Dim Test1 As Integer = 0
    Dim Holder(3) As Integer
    Holder(1) = Reference_Position
    While Test1 = 0
        If Val.Chars(Reference_Position + Step_Count) = "(" Then Left_Brace_Counter = Left_Brace_Counter + 1
        If Val.Chars(Reference_Position + Step_Count) = ")" Then Right_Brace_Counter = Right_Brace_Counter + 1
        If Left_Brace_Counter = Right_Brace_Counter Then
            Holder(3) = Reference_Position + Step_Count
            Test1 = 1
        End If
        Step_Count = Step_Count + 1
    End While
    Test1 = 0
    Step_Count = Holder(1)
    Left_Brace_Counter = 0
    Right_Brace_Counter = 0
    While Test1 = 0
        If Val.Chars(Step_Count) = "(" Then Left_Brace_Counter = Left_Brace_Counter + 1
        If Val.Chars(Step_Count) = ")" Then Right_Brace_Counter = Right_Brace_Counter + 1
        If Left_Brace_Counter = Right_Brace_Counter + 1 And (Val.Chars(Step_Count) = "+" Or Val.Chars(Step_Count) = "|") Then
            Holder(2) = Step_Count
            If Val.Chars(Step_Count) = "+" Then Holder(0) = 1
            Test1 = 1
        End If
        Step_Count = Step_Count + 1
    End While
    Return Holder
End Function

Public Function right_subwff(ByVal Val As String, ByVal Reference_Position As Integer) As Array
    Dim Left_Brace_Counter As Integer
    Dim Right_Brace_Counter As Integer
    Dim Step_Count As Integer
    Dim Test1 As Integer
    Dim Holder(3) As Integer
    Holder(1) = Reference_Position
    If Val.Chars(Reference_Position) = "(" Then
        While Test1 = 0
            If Val.Chars(Reference_Position + Step_Count) = "(" Then Left_Brace_Counter = Left_Brace_Counter + 1
            If Val.Chars(Reference_Position + Step_Count) = ")" Then Right_Brace_Counter = Right_Brace_Counter + 1
            If Left_Brace_Counter = Right_Brace_Counter Then
                Holder(3) = Reference_Position + Step_Count
                Test1 = 1
            End If
            Step_Count = Step_Count + 1
        End While
    Else
        While Test1 = 0
            If Not (UCase(Val.Chars(Reference_Position + Step_Count)) = LCase(Val.Chars(Reference_Position + Step_Count))) Then
                Holder(3) = Reference_Position + Step_Count
                Test1 = 1
            End If
            Step_Count = Step_Count + 1
        End While
    End If
    Return Holder
End Function

Public Function left_subwff(ByVal Val As String, ByVal Reference_Position As Integer) As Array
    Dim Left_Brace_Counter As Integer
    Dim Right_Brace_Counter As Integer
    Dim Step_Count As Integer
    Dim Test1 As Integer
    Dim Holder(3) As Integer
    Holder(3) = Reference_Position
    If Val.Chars(Reference_Position) = ")" Then
        While Test1 = 0
            If Val.Chars(Reference_Position - Step_Count) = ")" Then Right_Brace_Counter = Right_Brace_Counter + 1
            If Val.Chars(Reference_Position - Step_Count) = "(" Then Left_Brace_Counter = Left_Brace_Counter + 1
            If Left_Brace_Counter = Right_Brace_Counter Then
                Holder(1) = Reference_Position - Step_Count
                Test1 = 1
            End If
            Step_Count = Step_Count + 1
        End While
    Else
        While Test1 = 0
            If Val.Chars(Reference_Position - (Step_Count + 1)) = "(" Then
                Holder(1) = Reference_Position - Step_Count
                Test1 = 1
            End If
            Step_Count = Step_Count + 1
        End While
    End If
    Return Holder
End Function

Public Function simplify_CNF1(ByVal Val As String) As String
    Dim Val_Copy As String = "("
    Dim Test1 As Integer = 0
    Dim Test2 As Integer
    Dim Step_Counter As Integer
```

```
    Test2 = 0
    Step_Counter = 0
    While Step_Counter < Val.Length
        If Val.Chars(Step_Counter) = "~" And Step_Counter <= Val.Length - 1 Then
            If Not (UCase(Val.Chars(Step_Counter + 1)) = LCase(Val.Chars(Step_Counter + 1))) Then
                Val_Copy = Val_Copy & "~" & Val.Chars(Step_Counter + 1)
                Test2 = 1
            End If
        End If
        If Not (UCase(Val.Chars(Step_Counter)) = LCase(Val.Chars(Step_Counter))) Then
            Val_Copy = Val_Copy & Val.Chars(Step_Counter)
        End If
        If Val.Chars(Step_Counter) = "|" Then
            Val_Copy = Val_Copy & "|"
        End If
        If Val.Chars(Step_Counter) = "+" Then
            Val_Copy = Val_Copy & ")+("
        End If
        Step_Counter = Step_Counter + 1
        If Test2 = 1 Then
            Test2 = 0
            Step_Counter = Step_Counter + 1
        End If
    End While
    Val_Copy = Val_Copy & ")"
    Return Val_Copy
End Function

Private Function return_probability(ByVal Input_String As String) As Double
    Dim output As Double = 0
    Dim Input_Cons As String = ""
    Dim String_Holder As String = ""
    Dim Input_Ante As String = ""
    Dim Input_Num_Val As Double = 0
    Dim Input_Den_Val As Double = 0
    Dim Test1 As Integer
    Dim Test2 As Integer = 0
    Dim Counter1 As Integer
    Dim Testa As Integer
    Dim apos As Integer = 99
    Dim Testb As Integer
    Dim bpos As Integer = 99
    Dim Testc As Integer
    Dim cpos As Integer = 99
    Dim Testd As Integer
    Dim dpos As Integer = 99
    Dim holder As Integer
    Dim Nega As Integer = 0
    Dim Negb As Integer = 0
    Dim Negc As Integer = 0
    Dim Negd As Integer = 0

    Input_String = Input_String & "*"

    Counter1 = 1
    While Test1 = 0
        If Input_String.Chars(Counter1) = "a" And Input_String.Chars(Counter1 - 1) = "~" Then Nega = 1
        If Input_String.Chars(Counter1) = "b" And Input_String.Chars(Counter1 - 1) = "~" Then Negb = 1
        If Input_String.Chars(Counter1) = "c" And Input_String.Chars(Counter1 - 1) = "~" Then Negc = 1
        If Input_String.Chars(Counter1) = "d" And Input_String.Chars(Counter1 - 1) = "~" Then Negd = 1
        If Input_String.Chars(Counter1) = "*" Then Test1 = 1
        Counter1 = Counter1 + 1
    End While

    If Not UCase(Input_String.Chars(0)) = LCase(Input_String.Chars(0)) Then
        Input_Ante = Input_String.Chars(0)
        Input_Cons = Input_String.Chars(0)
        Test2 = 1
    End If

    Counter1 = 1
    Test1 = 0
    While Test1 = 0
        If Not UCase(Input_String.Chars(Counter1)) = LCase(Input_String.Chars(Counter1)) Then
            If Test2 = 1 Then
                Input_Ante = Input_Ante & "+" & Input_String.Chars(Counter1)
                Input_Cons = Input_Cons & "+" & Input_String.Chars(Counter1)
            Else
                Input_Ante = Input_Ante & Input_String.Chars(Counter1)
                Input_Cons = Input_Cons & Input_String.Chars(Counter1)
                Test2 = 1
            End If
        End If
        If Input_String.Chars(Counter1) = "-" Then Test1 = 1
        Counter1 = Counter1 + 1
    End While

    Test1 = 0
    While Test1 = 0
        If Not UCase(Input_String.Chars(Counter1)) = LCase(Input_String.Chars(Counter1)) Then
            Input_Cons = Input_Cons & "+" & Input_String.Chars(Counter1)
        End If
        If Input_String.Chars(Counter1 + 1) = "*" Then Test1 = 1
        Counter1 = Counter1 + 1
    End While

    String_Holder = Input_Cons & "*"
    Counter1 = 0
    Test1 = 0
    While Test1 = 0
        If String_Holder.Chars(Counter1) = "a" Then
            Testa = 1
            apos = Counter1
        End If
        If String_Holder.Chars(Counter1) = "b" Then
            Testb = 1
            bpos = Counter1
        End If
        If String_Holder.Chars(Counter1) = "c" Then
            Testc = 1
            cpos = Counter1
        End If
        If String_Holder.Chars(Counter1) = "d" Then
            Testd = 1
            dpos = Counter1
        End If
        If String_Holder.Chars(Counter1 + 1) = "*" Then
            Test1 = 1
        End If
        Counter1 = Counter1 + 1
    End While

    Test1 = 0
    While Test1 = 0
        Test1 = 1
        If apos > bpos And Testa = 1 And Testb = 1 Then
            Input_Cons = Input_Cons.Replace("b", "*")
```

```
            Input_Cons = Input_Cons.Replace("a", "b")
            Input_Cons = Input_Cons.Replace("*", "a")
            holder = apos
            apos = bpos
            bpos = holder
            Test1 = 0
        End If
        If apos > cpos And Testa = 1 And Testc = 1 Then
            Input_Cons = Input_Cons.Replace("c", "*")
            Input_Cons = Input_Cons.Replace("a", "c")
            Input_Cons = Input_Cons.Replace("*", "a")
            holder = apos
            apos = cpos
            cpos = holder
            Test1 = 0
        End If
        If apos > dpos And Testa = 1 And Testd = 1 Then
            Input_Cons = Input_Cons.Replace("d", "*")
            Input_Cons = Input_Cons.Replace("a", "d")
            Input_Cons = Input_Cons.Replace("*", "a")
            holder = apos
            apos = dpos
            dpos = holder
            Test1 = 0
        End If
        If bpos > cpos And Testb = 1 And Testc = 1 Then
            Input_Cons = Input_Cons.Replace("c", "*")
            Input_Cons = Input_Cons.Replace("b", "c")
            Input_Cons = Input_Cons.Replace("*", "b")
            holder = bpos
            bpos = cpos
            cpos = holder
            Test1 = 0
        End If
        If bpos > dpos And Testb = 1 And Testd = 1 Then
            Input_Cons = Input_Cons.Replace("d", "*")
            Input_Cons = Input_Cons.Replace("b", "d")
            Input_Cons = Input_Cons.Replace("*", "b")
            holder = bpos
            bpos = dpos
            dpos = holder
            Test1 = 0
        End If
        If cpos > dpos And Testc = 1 And Testd = 1 Then
            Input_Cons = Input_Cons.Replace("d", "*")
            Input_Cons = Input_Cons.Replace("c", "d")
            Input_Cons = Input_Cons.Replace("*", "c")
            holder = cpos
            cpos = dpos
            dpos = holder
            Test1 = 0
        End If
    End While

    String_Holder = Input_Ante & "*"
    Counter1 = 0
    Test1 = 0
    While Test1 = 0
        If String_Holder.Chars(Counter1) = "a" And Nega = 1 Then
            Input_Ante = Input_Ante.Replace("a", "~a")
        End If
        If String_Holder.Chars(Counter1) = "b" And Negb = 1 Then
            Input_Ante = Input_Ante.Replace("b", "~b")
        End If
        If String_Holder.Chars(Counter1) = "c" And Negc = 1 Then
            Input_Ante = Input_Ante.Replace("c", "~c")
        End If
        If String_Holder.Chars(Counter1) = "d" And Negd = 1 Then
            Input_Ante = Input_Ante.Replace("d", "~d")
        End If
        If String_Holder.Chars(Counter1) = "*" Then Test1 = 1
        Counter1 = Counter1 + 1
    End While

    String_Holder = Input_Cons & "*"
    Counter1 = 0
    Test1 = 0
    While Test1 = 0
        If String_Holder.Chars(Counter1) = "a" And Nega = 1 Then
            Input_Cons = Input_Cons.Replace("a", "~a")
        End If
        If String_Holder.Chars(Counter1) = "b" And Negb = 1 Then
            Input_Cons = Input_Cons.Replace("b", "~b")
        End If
        If String_Holder.Chars(Counter1) = "c" And Negc = 1 Then
            Input_Cons = Input_Cons.Replace("c", "~c")
        End If
        If String_Holder.Chars(Counter1) = "d" And Negd = 1 Then
            Input_Cons = Input_Cons.Replace("d", "~d")
        End If
        If String_Holder.Chars(Counter1) = "*" Then Test1 = 1
        Counter1 = Counter1 + 1
    End While

    For x = 16 To 95
        If Input_Cons = Proposition(x) Then Input_Num_Val = Probability(x)
        If Input_Ante = Proposition(x) Then Input_Den_Val = Probability(x)
    Next

    Return Input_Num_Val / Input_Den_Val

End Function

Private Sub load_consequence_sets()
    Dim Counter As Integer


    Counter = 1
    For x = 1 To Number_of_Props_Queried
        If Z_Judgments(x) = 1 Then
            Z_Conclusions(Counter) = Queried_Propositions(x)
            Counter = Counter + 1
        End If
    Next

    Counter = 1
    For x = 1 To Number_of_Props_Queried
        If Zs_Judgments(x) = 1 Then
            Zs_Conclusions(Counter) = Queried_Propositions(x)
            Counter = Counter + 1
        End If
    Next

    Counter = 1
    For x = 1 To Number_of_Props_Queried
        If P_Judgments(x) = 1 Then
            P_Conclusions(Counter) = Queried_Propositions(x)
```

```
            Counter = Counter + 1
        End If
    Next

    Counter = 1
    For x = 1 To Number_of_Props_Queried
        If Classical_Judgments(x) = 1 Then
            Classical_Conclusions(Counter) = Queried_Propositions(x)
            Counter = Counter + 1
        End If
    Next

End Sub

Private Function trans_PProposition_to_Proposition(ByVal Input As String) As String
    Dim Test5 As Integer
    Dim Output As String = ""
    Dim Ante_Lit_Count As Integer = 0
    Dim Cons_Lit_Count As Integer = 0
    Dim String_Position As Integer = 0
    Dim Cons_Length As Integer
    Dim Ante As String = ""
    Dim Cons As String = ""

    Test5 = 0
    While Test5 = 0
        If Not (UCase(Input.Chars(String_Position)) = LCase(Input.Chars(String_Position))) Then Cons_Lit_Count = Cons_Lit_Count + 1
        If Input.Chars(String_Position) = "|" Then Test5 = 1
        String_Position = String_Position + 1
    End While
    Test5 = 0
    While Test5 = 0
        If Not (UCase(Input.Chars(String_Position)) = LCase(Input.Chars(String_Position))) Then Ante_Lit_Count = Ante_Lit_Count + 1
        If Input.Length = String_Position + 1 Then Test5 = 1
        String_Position = String_Position + 1
    End While


    If Cons_Lit_Count = 1 Then
        If Not UCase(Input.Chars(0)) = LCase(Input.Chars(0)) Then
            Cons = Input.Chars(0)
            Cons_Length = 2
        Else
            Cons = "~" & Input.Chars(1)
            Cons_Length = 3
        End If
    End If

    If Cons_Lit_Count = 2 Then
        If Not UCase(Input.Chars(0)) = LCase(Input.Chars(0)) Then
            If Not UCase(Input.Chars(2)) = LCase(Input.Chars(2)) Then
                Cons = "(" & Input.Chars(0) & "+" & Input.Chars(2) & ")"
                Cons_Length = 4
            Else
                Cons = "(" & Input.Chars(0) & "+~" & Input.Chars(3) & ")"
                Cons_Length = 5
            End If
        Else
            If Not UCase(Input.Chars(3)) = LCase(Input.Chars(3)) Then
                Cons = "(~" & Input.Chars(1) & "+" & Input.Chars(3) & ")"
                Cons_Length = 5
            Else
                Cons = "(~" & Input.Chars(1) & "+~" & Input.Chars(4) & ")"
                Cons_Length = 6
            End If
        End If
    End If

    If Cons_Lit_Count = 3 Then
        If Not UCase(Input.Chars(0)) = LCase(Input.Chars(0)) Then
            If Not UCase(Input.Chars(2)) = LCase(Input.Chars(2)) Then
                If Not UCase(Input.Chars(4)) = LCase(Input.Chars(4)) Then
                    Cons = "((" & Input.Chars(0) & "+" & Input.Chars(2) & ")+" & Input.Chars(4) & ")"
                    Cons_Length = 6
                Else
                    Cons = "((" & Input.Chars(0) & "+" & Input.Chars(2) & ")+~" & Input.Chars(5) & ")"
                    Cons_Length = 7
                End If
            Else
                If Not UCase(Input.Chars(5)) = LCase(Input.Chars(5)) Then
                    Cons = "((" & Input.Chars(0) & "+~" & Input.Chars(3) & ")+" & Input.Chars(5) & ")"
                    Cons_Length = 7
                Else
                    Cons = "((" & Input.Chars(0) & "+~" & Input.Chars(3) & ")+~" & Input.Chars(6) & ")"
                    Cons_Length = 8
                End If
            End If
        Else
            If Not UCase(Input.Chars(3)) = LCase(Input.Chars(3)) Then
                If Not UCase(Input.Chars(5)) = LCase(Input.Chars(5)) Then
                    Cons = "((~" & Input.Chars(1) & "+" & Input.Chars(3) & ")+" & Input.Chars(5) & ")"
                    Cons_Length = 7
                Else
                    Cons = "((~" & Input.Chars(1) & "+" & Input.Chars(3) & ")+~" & Input.Chars(6) & ")"
                    Cons_Length = 8
                End If
            Else
                If Not UCase(Input.Chars(6)) = LCase(Input.Chars(6)) Then
                    Cons = "((~" & Input.Chars(1) & "+~" & Input.Chars(4) & ")+" & Input.Chars(6) & ")"
                    Cons_Length = 8
                Else
                    Cons = "((~" & Input.Chars(1) & "+~" & Input.Chars(4) & ")+~" & Input.Chars(7) & ")"
                    Cons_Length = 9
                End If
            End If
        End If
    End If

    If Ante_Lit_Count = 1 Then
        If Not UCase(Input.Chars(Cons_Length)) = LCase(Input.Chars(Cons_Length)) Then
            Ante = Input.Chars(Cons_Length)
        Else
            Ante = "~" & Input.Chars(Cons_Length + 1)
        End If
    End If

    If Ante_Lit_Count = 2 Then
        If Not UCase(Input.Chars(Cons_Length)) = LCase(Input.Chars(Cons_Length)) Then
            If Not UCase(Input.Chars(Cons_Length + 2)) = LCase(Input.Chars(Cons_Length + 2)) Then
                Ante = "(" & Input.Chars(Cons_Length) & "+" & Input.Chars(Cons_Length + 2) & ")"
            Else
                Ante = "(" & Input.Chars(Cons_Length) & "+~" & Input.Chars(Cons_Length + 3) & ")"
            End If
        Else
            If Not UCase(Input.Chars(Cons_Length + 3)) = LCase(Input.Chars(Cons_Length + 3)) Then
                Ante = "(~" & Input.Chars(Cons_Length + 1) & "+" & Input.Chars(Cons_Length + 3) & ")"
            Else
```

```
                Ante = "(~" & Input.Chars(Cons_Length + 1) & "+~" & Input.Chars(Cons_Length + 4) & ")"
            End If
        End If
    End If

    If Ante_Lit_Count = 3 Then
        If Not UCase(Input.Chars(Cons_Length)) = LCase(Input.Chars(Cons_Length)) Then
            If Not UCase(Input.Chars(Cons_Length + 2)) = LCase(Input.Chars(Cons_Length + 2)) Then
                If Not UCase(Input.Chars(Cons_Length + 4)) = LCase(Input.Chars(Cons_Length + 4)) Then
                    Ante = "((" & Input.Chars(Cons_Length) & "+" & Input.Chars(Cons_Length + 2) & ")+" & Input.Chars(Cons_Length + 4) & ")"
                Else
                    Ante = "((" & Input.Chars(Cons_Length) & "+" & Input.Chars(Cons_Length + 2) & ")+~" & Input.Chars(Cons_Length + 5) & ")"
                End If
            Else
                If Not UCase(Input.Chars(Cons_Length + 5)) = LCase(Input.Chars(Cons_Length + 5)) Then
                    Ante = "((" & Input.Chars(Cons_Length) & "+~" & Input.Chars(Cons_Length + 3) & ")+" & Input.Chars(Cons_Length + 5) & ")"
                Else
                    Ante = "((" & Input.Chars(Cons_Length) & "+~" & Input.Chars(Cons_Length + 3) & ")+~" & Input.Chars(Cons_Length + 6) & ")"
                End If
            End If
        Else
            If Not UCase(Input.Chars(Cons_Length + 3)) = LCase(Input.Chars(Cons_Length + 3)) Then
                If Not UCase(Input.Chars(Cons_Length + 5)) = LCase(Input.Chars(Cons_Length + 5)) Then
                    Ante = "((~" & Input.Chars(Cons_Length + 1) & "+" & Input.Chars(Cons_Length + 3) & ")+" & Input.Chars(Cons_Length + 5) & ")"
                Else
                    Ante = "((~" & Input.Chars(Cons_Length + 1) & "+" & Input.Chars(Cons_Length + 3) & ")+~" & Input.Chars(Cons_Length + 6) & ")"
                End If
            Else
                If Not UCase(Input.Chars(Cons_Length + 6)) = LCase(Input.Chars(Cons_Length + 6)) Then
                    Ante = "((~" & Input.Chars(Cons_Length + 1) & "+~" & Input.Chars(Cons_Length + 4) & ")+" & Input.Chars(Cons_Length + 6) & ")"
                Else
                    Ante = "((~" & Input.Chars(Cons_Length + 1) & "+~" & Input.Chars(Cons_Length + 4) & ")+~" & Input.Chars(Cons_Length + 7) & ")"
                End If
            End If
        End If
    End If

    Output = Ante & "-" & Cons

    Return Output
End Function

Private Function get_value() As Double 'probabilistic distribuion of three cases according to initial prob.
    RandomNumber = RandomClass.NextDouble()
    Return RandomNumber
End Function

'Private Sub gererate_queries()
'    Dim testx As Integer
'    For x = 1 To Number_of_Props_Queried
'        testx = 1
'        While testx = 1
'            testx = 0
'            gererate_datum()
'            For y = 1 To x - 1 'loop determines whether datum was alread selected.
'                If Queried_PPropositions(y) = Datum Then testx = 1
'            Next
'            If testx = 0 Then
'                Queried_PPropositions(x) = Datum
'                Queried_Probabilities(x) = Datum_Prob
'                Queried_Propositions(x) = trans_PProposition_to_Proposition(Queried_PPropositions(x))
'            End If
'        End While
'    Next
'End Sub

Private Sub gererate_all_sensible_conds()
    Dim Holder_Array(2, 4) As String
    Dim Counter As Integer = 1
    Dim Input_Ante As String = ""
    Dim Input_Cons As String = ""
    Dim Test6 As Integer
    Dim String_Position As Integer
    Dim Input_Num As String = ""
    Dim Input_Den As String = ""
    Dim Input_Num_Val As Double = 0
    Dim Input_Den_Val As Double = 0

    For x1 = 1 To 3
        For x2 = 1 To 3
            For x3 = 1 To 3
                For x4 = 1 To 3
                    For x5 = 1 To 3
                        For x6 = 1 To 3
                            For x7 = 1 To 3
                                For x8 = 1 To 3

                                    If x1 = 1 And x2 = 1 And x3 = 1 And x4 = 3 And x5 = 2 And x6 = 3 And x7 = 2 And x8 = 1 Then
                                        x1 = x1
                                    End If

                                    Select Case x1
                                        Case 1
                                            Holder_Array(1, 1) = "a"
                                        Case 2
                                            Holder_Array(1, 1) = "~a"
                                    End Select

                                    Select Case x2
                                        Case 1
                                            Holder_Array(1, 2) = "b"
                                        Case 2
                                            Holder_Array(1, 2) = "~b"
                                    End Select

                                    Select Case x3
                                        Case 1
                                            Holder_Array(1, 3) = "c"
                                        Case 2
                                            Holder_Array(1, 3) = "~c"
                                    End Select

                                    Select Case x4
                                        Case 1
                                            Holder_Array(1, 4) = "d"
                                        Case 2
                                            Holder_Array(1, 4) = "~d"
                                    End Select

                                    Select Case x5
                                        Case 1
                                            Holder_Array(2, 1) = "a"
                                        Case 2
                                            Holder_Array(2, 1) = "~a"
                                    End Select

                                    Select Case x6
```

```
                                    Case 1
                                        Holder_Array(2, 2) = "b"
                                    Case 2
                                        Holder_Array(2, 2) = "~b"
                                End Select

                                Select Case x7
                                    Case 1
                                        Holder_Array(2, 3) = "c"
                                    Case 2
                                        Holder_Array(2, 3) = "~c"
                                End Select

                                Select Case x8
                                    Case 1
                                        Holder_Array(2, 4) = "d"
                                    Case 2
                                        Holder_Array(2, 4) = "~d"
                                End Select

                                Test6 = 1

                                For x = 1 To 4
                                    If Holder_Array(1, x) = Holder_Array(2, x) Or "~" & Holder_Array(1, x) = Holder_Array(2, x) Or Holder_Array(1, x) = "~" &
Holder_Array(2, x) Then

                                        Holder_Array(1, x) = ""
                                        Holder_Array(2, x) = ""
                                    End If
                                Next

                                Input_Cons = Holder_Array(1, 1) & Holder_Array(1, 2) & Holder_Array(1, 3) & Holder_Array(1, 4)
                                Input_Ante = Holder_Array(2, 1) & Holder_Array(2, 2) & Holder_Array(2, 3) & Holder_Array(2, 4)

                                If Input_Cons = "" Or Input_Ante = "" Then Test6 = 0


                                If Test6 = 1 Then
                                    Test6 = 0

                                    String_Position = 0
                                    Test6 = 0
                                    While Test6 = 0
                                        If Not (UCase(Input_Cons.Chars(String_Position)) = LCase(Input_Cons.Chars(String_Position))) And Not (Input_Cons.Length =
String_Position + 1) Then

                                            Input_Cons = Input_Cons.Replace(Input_Cons.Chars(String_Position), Input_Cons.Chars(String_Position) & "+")
                                        End If
                                        If Input_Cons.Length = String_Position + 1 Then Test6 = 1
                                        String_Position = String_Position + 1
                                    End While

                                    String_Position = 0
                                    Test6 = 0
                                    While Test6 = 0
                                        If Not (UCase(Input_Ante.Chars(String_Position)) = LCase(Input_Ante.Chars(String_Position))) And Not (Input_Ante.Length =
String_Position + 1) Then

                                            Input_Ante = Input_Ante.Replace(Input_Ante.Chars(String_Position), Input_Ante.Chars(String_Position) & "+")
                                        End If
                                        If Input_Ante.Length = String_Position + 1 Then Test6 = 1
                                        String_Position = String_Position + 1
                                    End While

                                    For x = 1 To 4
                                        If Holder_Array(1, x) = "" Then Holder_Array(1, x) = Holder_Array(2, x)
                                    Next

                                    Input_Num = Holder_Array(1, 1) & Holder_Array(1, 2) & Holder_Array(1, 3) & Holder_Array(1, 4)
                                    Input_Den = Holder_Array(2, 1) & Holder_Array(2, 2) & Holder_Array(2, 3) & Holder_Array(2, 4)

                                    String_Position = 0
                                    Test6 = 0
                                    While Test6 = 0
                                        If Not (UCase(Input_Num.Chars(String_Position)) = LCase(Input_Num.Chars(String_Position))) And Not (Input_Num.Length =
String_Position + 1) Then

                                            Input_Num = Input_Num.Replace(Input_Num.Chars(String_Position), Input_Num.Chars(String_Position) & "+")
                                        End If
                                        If Input_Num.Length = String_Position + 1 Then Test6 = 1
                                        String_Position = String_Position + 1
                                    End While

                                    String_Position = 0
                                    Test6 = 0
                                    While Test6 = 0
                                        If Not (UCase(Input_Den.Chars(String_Position)) = LCase(Input_Den.Chars(String_Position))) And Not (Input_Den.Length =
String_Position + 1) Then

                                            Input_Den = Input_Den.Replace(Input_Den.Chars(String_Position), Input_Den.Chars(String_Position) & "+")
                                        End If
                                        If Input_Den.Length = String_Position + 1 Then Test6 = 1
                                        String_Position = String_Position + 1
                                    End While

                                    For x = 16 To 95
                                        If Input_Num = Proposition(x) Then Input_Num_Val = Probability(x)
                                        If Input_Den = Proposition(x) Then Input_Den_Val = Probability(x)
                                    Next

                                    If Counter = 1 Then
                                        Counter = Counter
                                    End If

                                    Test6 = 1
                                    For y = 1 To Counter - 1
                                        If Queried_PPropositions(y) = Input_Cons & "|" & Input_Ante Then Test6 = 0
                                    Next

                                    If Test6 = 1 Then
                                        Queried_PPropositions(Counter) = Input_Cons & "|" & Input_Ante
                                        Queried_Probabilities(Counter) = Input_Num_Val / Input_Den_Val
                                        Queried_Propositions(Counter) = trans_PProposition_to_Proposition(Queried_PPropositions(Counter))
                                        Counter = Counter + 1
                                    End If

                                    For x = 1 To 2
                                        For y = 1 To 4
                                            Holder_Array(x, y) = ""
                                        Next
                                    Next

                                End If

                            Next
                        Next
                    Next
                Next
            Next
        Next
    Next
Next
```

```
End Sub

Private Sub gererate_datum()
    Dim Test6 As Integer
    Dim Holder As Double
    Dim Holder_Array(2, 4) As String
    Dim Input_Ante As String = ""
    Dim Input_Cons As String = ""
    Dim Input_Num As String = ""
    Dim Input_Den As String = ""
    Dim Input_Num_Val As Double = 0
    Dim Input_Den_Val As Double = 0
    Dim String_Position As Integer = 1

    Test6 = 0
    While Test6 = 0
        Test6 = 1

        Holder = get_value()
        If Holder <= 0.33333 Then Holder_Array(1, 1) = "~a"
        If Holder >= 0.6666 Then Holder_Array(1, 1) = "a"
        Holder = get_value()
        If Holder <= 0.33333 Then Holder_Array(1, 2) = "~b"
        If Holder >= 0.66666 Then Holder_Array(1, 2) = "b"
        Holder = get_value()
        If Holder <= 0.33333 Then Holder_Array(1, 3) = "~c"
        If Holder >= 0.66666 Then Holder_Array(1, 3) = "c"
        Holder = get_value()
        If Holder <= 0.33333 Then Holder_Array(1, 4) = "~d"
        If Holder >= 0.66666 Then Holder_Array(1, 4) = "d"

        Holder = get_value()
        If Holder <= 0.33333 Then Holder_Array(2, 1) = "~a"
        If Holder >= 0.66666 Then Holder_Array(2, 1) = "a"
        Holder = get_value()
        If Holder <= 0.33333 Then Holder_Array(2, 2) = "~b"
        If Holder >= 0.66666 Then Holder_Array(2, 2) = "b"
        Holder = get_value()
        If Holder <= 0.33333 Then Holder_Array(2, 3) = "~c"
        If Holder >= 0.66666 Then Holder_Array(2, 3) = "c"
        Holder = get_value()
        If Holder <= 0.33333 Then Holder_Array(2, 4) = "~d"
        If Holder >= 0.66666 Then Holder_Array(2, 4) = "d"

        For x = 1 To 4
            If Holder_Array(1, x) = Holder_Array(2, x) Or "~" & Holder_Array(1, x) = Holder_Array(2, x) Or Holder_Array(1, x) = "~" & Holder_Array(2, x) Then
                Holder_Array(1, x) = ""
                Holder_Array(2, x) = ""
            End If
        Next

        Input_Cons = Holder_Array(1, 1) & Holder_Array(1, 2) & Holder_Array(1, 3) & Holder_Array(1, 4)
        Input_Ante = Holder_Array(2, 1) & Holder_Array(2, 2) & Holder_Array(2, 3) & Holder_Array(2, 4)

        If Input_Cons = "" Or Input_Ante = "" Then Test6 = 0

        If Test6 = 0 Then
            For x = 1 To 2
                For y = 1 To 4
                    Holder_Array(x, y) = ""
                Next
            Next
        End If

    End While

    String_Position = 0
    Test6 = 0
    While Test6 = 0
        If Not (UCase(Input_Cons.Chars(String_Position)) = LCase(Input_Cons.Chars(String_Position))) And Not (Input_Cons.Length = String_Position + 1) Then
            Input_Cons = Input_Cons.Replace(Input_Cons.Chars(String_Position), Input_Cons.Chars(String_Position) & "+")
        End If
        If Input_Cons.Length = String_Position + 1 Then Test6 = 1
        String_Position = String_Position + 1
    End While

    String_Position = 0
    Test6 = 0
    While Test6 = 0
        If Not (UCase(Input_Ante.Chars(String_Position)) = LCase(Input_Ante.Chars(String_Position))) And Not (Input_Ante.Length = String_Position + 1) Then
            Input_Ante = Input_Ante.Replace(Input_Ante.Chars(String_Position), Input_Ante.Chars(String_Position) & "+")
        End If
        If Input_Ante.Length = String_Position + 1 Then Test6 = 1
        String_Position = String_Position + 1
    End While

    Datum = Input_Cons & "|" & Input_Ante

    For x = 1 To 4
        If Holder_Array(1, x) = "" Then Holder_Array(1, x) = Holder_Array(2, x)
    Next

    Input_Num = Holder_Array(1, 1) & Holder_Array(1, 2) & Holder_Array(1, 3) & Holder_Array(1, 4)
    Input_Den = Holder_Array(2, 1) & Holder_Array(2, 2) & Holder_Array(2, 3) & Holder_Array(2, 4)

    String_Position = 0
    Test6 = 0
    While Test6 = 0
        If Not (UCase(Input_Num.Chars(String_Position)) = LCase(Input_Num.Chars(String_Position))) And Not (Input_Num.Length = String_Position + 1) Then
            Input_Num = Input_Num.Replace(Input_Num.Chars(String_Position), Input_Num.Chars(String_Position) & "+")
        End If
        If Input_Num.Length = String_Position + 1 Then Test6 = 1
        String_Position = String_Position + 1
    End While

    String_Position = 0
    Test6 = 0
    While Test6 = 0
        If Not (UCase(Input_Den.Chars(String_Position)) = LCase(Input_Den.Chars(String_Position))) And Not (Input_Den.Length = String_Position + 1) Then
            Input_Den = Input_Den.Replace(Input_Den.Chars(String_Position), Input_Den.Chars(String_Position) & "+")
        End If
        If Input_Den.Length = String_Position + 1 Then Test6 = 1
        String_Position = String_Position + 1
    End While

    For x = 16 To 95
        If Input_Num = Proposition(x) Then Input_Num_Val = Probability(x)
        If Input_Den = Proposition(x) Then Input_Den_Val = Probability(x)
    Next

    Datum_Prob = Input_Num_Val / Input_Den_Val

End Sub

Private Function generate_P_Value(ByVal Input As String) As Double
    Dim Test7 As Integer = 0
```

```
Dim Test8 As Integer = 0
Dim String_Position As Integer = 1
Dim Holder_Array(2, 4) As String
Dim Input_Num As String
Dim Input_Den As String
Dim Input_Num_Val As Double = 0
Dim Input_Den_Val As Double = 0
Dim Output As Double

Select Case Input.Chars(0)
    Case "a"
        Holder_Array(1, 1) = "a"
    Case "b"
        Holder_Array(1, 2) = "b"
    Case "c"
        Holder_Array(1, 3) = "c"
    Case "d"
        Holder_Array(1, 4) = "d"
End Select

Test8 = 0
While Test8 = 0
    If Input.Chars(String_Position) = "a" Then
        If Input.Chars(String_Position - 1) = "~" Then
            Holder_Array(1, 1) = "~a"
        Else
            Holder_Array(1, 1) = "a"
        End If
    End If
    If Input.Chars(String_Position) = "b" Then
        If Input.Chars(String_Position - 1) = "~" Then
            Holder_Array(1, 2) = "~b"
        Else
            Holder_Array(1, 2) = "b"
        End If
    End If
    If Input.Chars(String_Position) = "c" Then
        If Input.Chars(String_Position - 1) = "~" Then
            Holder_Array(1, 3) = "~c"
        Else
            Holder_Array(1, 3) = "c"
        End If
    End If
    If Input.Chars(String_Position) = "d" Then
        If Input.Chars(String_Position - 1) = "~" Then
            Holder_Array(1, 4) = "~d"
        Else
            Holder_Array(1, 4) = "d"
        End If
    End If
    If Input.Chars(String_Position) = "|" Then Test8 = 1
    String_Position = String_Position + 1
End While

Test8 = 0
While Test8 = 0
    If Input.Chars(String_Position) = "a" Then
        If Input.Chars(String_Position - 1) = "~" Then
            If Holder_Array(1, 1) = "a" Then Test7 = 1
            Holder_Array(1, 1) = "~a"
            Holder_Array(2, 1) = "~a"
        Else
            If Holder_Array(2, 1) = "~a" Then Test7 = 1
            Holder_Array(1, 1) = "a"
            Holder_Array(2, 1) = "a"
        End If
    End If

    If Input.Chars(String_Position) = "b" Then
        If Input.Chars(String_Position - 1) = "~" Then
            If Holder_Array(1, 2) = "b" Then Test7 = 1
            Holder_Array(1, 2) = "~b"
            Holder_Array(2, 2) = "~b"
        Else
            If Holder_Array(1, 2) = "~b" Then Test7 = 1
            Holder_Array(1, 2) = "b"
            Holder_Array(2, 2) = "b"
        End If
    End If

    If Input.Chars(String_Position) = "c" Then
        If Input.Chars(String_Position - 1) = "~" Then
            If Holder_Array(1, 3) = "c" Then Test7 = 1
            Holder_Array(1, 3) = "~c"
            Holder_Array(2, 3) = "~c"
        Else
            If Holder_Array(1, 3) = "~c" Then Test7 = 1
            Holder_Array(1, 3) = "c"
            Holder_Array(2, 3) = "c"
        End If
    End If

    If Input.Chars(String_Position) = "d" Then
        If Input.Chars(String_Position - 1) = "~" Then
            If Holder_Array(1, 4) = "d" Then Test7 = 1
            Holder_Array(1, 4) = "~d"
            Holder_Array(2, 4) = "~d"
        Else
            If Holder_Array(1, 4) = "~d" Then Test7 = 1
            Holder_Array(1, 4) = "d"
            Holder_Array(2, 4) = "d"
        End If
    End If
    String_Position = String_Position + 1
    If Input.Length = String_Position Then Test8 = 1
End While

Input_Num = Holder_Array(1, 1) & Holder_Array(1, 2) & Holder_Array(1, 3) & Holder_Array(1, 4)
Input_Den = Holder_Array(2, 1) & Holder_Array(2, 2) & Holder_Array(2, 3) & Holder_Array(2, 4)

String_Position = 0
Test8 = 0
While Test8 = 0
    If Not (UCase(Input_Num.Chars(String_Position)) = LCase(Input_Num.Chars(String_Position))) And Not (Input_Num.Length = String_Position + 1) Then
        Input_Num = Input_Num.Replace(Input_Num.Chars(String_Position), Input_Num.Chars(String_Position) & "+")
    End If
    If Input_Num.Length = String_Position + 1 Then Test8 = 1
    String_Position = String_Position + 1
End While

String_Position = 0
Test8 = 0
While Test8 = 0
    If Not (UCase(Input_Den.Chars(String_Position)) = LCase(Input_Den.Chars(String_Position))) And Not (Input_Den.Length = String_Position + 1) Then
        Input_Den = Input_Den.Replace(Input_Den.Chars(String_Position), Input_Den.Chars(String_Position) & "+")
    End If
    If Input_Den.Length = String_Position + 1 Then Test8 = 1
```

```
            String_Position = String_Position + 1
        End While

        For x = 16 To 95
            If Input_Num = Proposition(x) Then Input_Num_Val = Probability(x)
            If Input_Den = Proposition(x) Then Input_Den_Val = Probability(x)
        Next

        Output = Input_Num_Val / Input_Den_Val
        If Test7 = 1 Then Output = 0
        Return Output

    End Function

    Private Sub load_P_Values()

        Proposition(16) = "a+b+c+d"
        Probability(16) = Probability(1) * Probability(2) * Probability(4) * Probability(8)
        Proposition(17) = "a+b+c+~d"
        Probability(17) = Probability(1) * Probability(2) * Probability(4) * (1 - Probability(8))
        Proposition(18) = "a+b+~c+d"
        Probability(18) = Probability(1) * Probability(2) * (1 - Probability(4)) * Probability(9)
        Proposition(19) = "a+b+~c+~d"
        Probability(19) = Probability(1) * Probability(2) * (1 - Probability(4)) * (1 - Probability(9))

        Proposition(20) = "a+~b+c+d"
        Probability(20) = Probability(1) * (1 - Probability(2)) * Probability(5) * Probability(10)
        Proposition(21) = "a+~b+c+~d"
        Probability(21) = Probability(1) * (1 - Probability(2)) * Probability(5) * (1 - Probability(10))
        Proposition(22) = "a+~b+~c+d"
        Probability(22) = Probability(1) * (1 - Probability(2)) * (1 - Probability(5)) * Probability(11)
        Proposition(23) = "a+~b+~c+~d"
        Probability(23) = Probability(1) * (1 - Probability(2)) * (1 - Probability(5)) * (1 - Probability(11))

        Proposition(24) = "~a+b+c+d"
        Probability(24) = (1 - Probability(1)) * Probability(3) * Probability(6) * Probability(12)
        Proposition(25) = "~a+b+c+~d"
        Probability(25) = (1 - Probability(1)) * Probability(3) * Probability(6) * (1 - Probability(12))
        Proposition(26) = "~a+b+~c+d"
        Probability(26) = (1 - Probability(1)) * Probability(3) * (1 - Probability(6)) * Probability(13)
        Proposition(27) = "~a+b+~c+~d"
        Probability(27) = (1 - Probability(1)) * Probability(3) * (1 - Probability(6)) * (1 - Probability(13))

        Proposition(28) = "~a+~b+c+d"
        Probability(28) = (1 - Probability(1)) * (1 - Probability(3)) * Probability(7) * Probability(14)
        Proposition(29) = "~a+~b+c+~d"
        Probability(29) = (1 - Probability(1)) * (1 - Probability(3)) * Probability(7) * (1 - Probability(14))
        Proposition(30) = "~a+~b+~c+d"
        Probability(30) = (1 - Probability(1)) * (1 - Probability(3)) * (1 - Probability(7)) * Probability(15)
        Proposition(31) = "~a+~b+~c+~d"
        Probability(31) = (1 - Probability(1)) * (1 - Probability(3)) * (1 - Probability(7)) * (1 - Probability(15))

        Proposition(32) = "a+b+c"
        Probability(32) = Probability(16) + Probability(17)
        Proposition(33) = "a+b+~c"
        Probability(33) = Probability(18) + Probability(19)
        Proposition(34) = "a+~b+c"
        Probability(34) = Probability(20) + Probability(21)
        Proposition(35) = "a+~b+~c"
        Probability(35) = Probability(22) + Probability(23)

        Proposition(36) = "~a+b+c"
        Probability(36) = Probability(24) + Probability(25)
        Proposition(37) = "~a+b+~c"
        Probability(37) = Probability(26) + Probability(27)
        Proposition(38) = "~a+~b+c"
        Probability(38) = Probability(28) + Probability(29)
        Proposition(39) = "~a+~b+~c"
        Probability(39) = Probability(30) + Probability(31)

        Proposition(40) = "a+b+d"
        Probability(40) = Probability(16) + Probability(18)
        Proposition(41) = "a+b+~d"
        Probability(41) = Probability(17) + Probability(19)
        Proposition(42) = "a+~b+d"
        Probability(42) = Probability(20) + Probability(22)
        Proposition(43) = "a+~b+~d"
        Probability(43) = Probability(21) + Probability(23)

        Proposition(44) = "~a+b+d"
        Probability(44) = Probability(24) + Probability(26)
        Proposition(45) = "~a+b+~d"
        Probability(45) = Probability(25) + Probability(27)
        Proposition(46) = "~a+~b+d"
        Probability(46) = Probability(28) + Probability(30)
        Proposition(47) = "~a+~b+~d"
        Probability(47) = Probability(29) + Probability(31)

        Proposition(48) = "a+c+d"
        Probability(48) = Probability(16) + Probability(20)
        Proposition(49) = "a+c+~d"
        Probability(49) = Probability(17) + Probability(21)
        Proposition(50) = "a+~c+d"
        Probability(50) = Probability(18) + Probability(22)
        Proposition(51) = "a+~c+~d"
        Probability(51) = Probability(19) + Probability(23)

        Proposition(52) = "~a+c+d"
        Probability(52) = Probability(24) + Probability(28)
        Proposition(53) = "~a+c+~d"
        Probability(53) = Probability(25) + Probability(29)
        Proposition(54) = "~a+~c+d"
        Probability(54) = Probability(26) + Probability(30)
        Proposition(55) = "~a+~c+~d"
        Probability(55) = Probability(27) + Probability(31)

        Proposition(56) = "b+c+d"
        Probability(56) = Probability(16) + Probability(24)
        Proposition(57) = "b+c+~d"
        Probability(57) = Probability(17) + Probability(25)
        Proposition(58) = "b+~c+d"
        Probability(58) = Probability(18) + Probability(26)
        Proposition(59) = "b+~c+~d"
        Probability(59) = Probability(19) + Probability(27)

        Proposition(60) = "~b+c+d"
        Probability(60) = Probability(20) + Probability(28)
        Proposition(61) = "~b+c+~d"
        Probability(61) = Probability(21) + Probability(29)
        Proposition(62) = "~b+~c+d"
        Probability(62) = Probability(22) + Probability(30)
        Proposition(63) = "~b+~c+~d"
        Probability(63) = Probability(23) + Probability(31)

        Proposition(64) = "a+b"
        Probability(64) = Probability(32) + Probability(33)
        Proposition(65) = "a+~b"
```

```
        Probability(65) = Probability(34) + Probability(35)
        Proposition(66) = "~a+b"
        Probability(66) = Probability(36) + Probability(37)
        Proposition(67) = "~a+~b"
        Probability(67) = Probability(38) + Probability(39)

        Proposition(68) = "a+c"
        Probability(68) = Probability(48) + Probability(49)
        Proposition(69) = "a+~c"
        Probability(69) = Probability(50) + Probability(51)
        Proposition(70) = "~a+c"
        Probability(70) = Probability(52) + Probability(53)
        Proposition(71) = "~a+~c"
        Probability(71) = Probability(54) + Probability(55)

        Proposition(72) = "a+d"
        Probability(72) = Probability(48) + Probability(50)
        Proposition(73) = "a+~d"
        Probability(73) = Probability(49) + Probability(51)
        Proposition(74) = "~a+d"
        Probability(74) = Probability(52) + Probability(54)
        Proposition(75) = "~a+~d"
        Probability(75) = Probability(53) + Probability(55)

        Proposition(76) = "b+c"
        Probability(76) = Probability(56) + Probability(57)
        Proposition(77) = "b+~c"
        Probability(77) = Probability(58) + Probability(59)
        Proposition(78) = "~b+c"
        Probability(78) = Probability(60) + Probability(61)
        Proposition(79) = "~b+~c"
        Probability(79) = Probability(62) + Probability(63)

        Proposition(80) = "b+d"
        Probability(80) = Probability(56) + Probability(58)
        Proposition(81) = "b+~d"
        Probability(81) = Probability(57) + Probability(59)
        Proposition(82) = "~b+d"
        Probability(82) = Probability(60) + Probability(62)
        Proposition(83) = "~b+~d"
        Probability(83) = Probability(61) + Probability(63)

        Proposition(84) = "c+d"
        Probability(84) = Probability(56) + Probability(60)
        Proposition(85) = "c+~d"
        Probability(85) = Probability(57) + Probability(61)
        Proposition(86) = "~c+d"
        Probability(86) = Probability(58) + Probability(62)
        Proposition(87) = "~c+~d"
        Probability(87) = Probability(59) + Probability(63)

        Proposition(88) = "a"
        Probability(88) = Probability(1)
        Proposition(89) = "~a"
        Probability(89) = 1 - Probability(1)
        Proposition(90) = "b"
        Probability(90) = Probability(64) + Probability(66)
        Proposition(91) = "~b"
        Probability(91) = 1 - Probability(90)
        Proposition(92) = "c"
        Probability(92) = Probability(84) + Probability(85)
        Proposition(93) = "~c"
        Probability(93) = 1 - Probability(92)
        Proposition(94) = "d"
        Probability(94) = Probability(84) + Probability(86)
        Proposition(95) = "~d"
        Probability(95) = 1 - Probability(94)

    End Sub

    Private Sub right_weakening()
        Dim Test1 As Integer
        Dim Main_Connective_Pointer As Integer
        Dim Pointer As Integer
        Dim Temp_Length As Integer = Number_of_O_Judgments
        Dim Size_of_Consequent As Integer
        Dim Consequent_Conj1 As Integer
        Dim Consequent_Conj2 As Integer
        Dim Candidate_Conclusions(6) As String

        For x = 1 To Temp_Length
            Main_Connective_Pointer = 0
            Test1 = 0

            While Test1 = 0
                If O_Conclusions(x).Chars(Main_Connective_Pointer) = "-" Then Test1 = 1
                Main_Connective_Pointer = Main_Connective_Pointer + 1
            End While

            Main_Connective_Pointer = Main_Connective_Pointer - 1
            Pointer = Main_Connective_Pointer + 1
            Size_of_Consequent = 0
            Test1 = 0

            While Test1 = 0
                If O_Conclusions(x).Chars(Pointer) = "+" Then
                    Size_of_Consequent = Size_of_Consequent + 1
                    If Size_of_Consequent = 1 Then Consequent_Conj1 = Pointer
                    If Size_of_Consequent = 2 Then Consequent_Conj2 = Pointer
                End If
                Pointer = Pointer + 1
                If Pointer = O_Conclusions(x).Length Then Test1 = 1
            End While

            If Size_of_Consequent = 1 Then
                Candidate_Conclusions(1) = O_Conclusions(x).Substring(0, Main_Connective_Pointer + 1) & O_Conclusions(x).Substring(Main_Connective_Pointer + 2, Consequent_Conj1 -
(Main_Connective_Pointer + 2))
                Candidate_Conclusions(2) = O_Conclusions(x).Substring(0, Main_Connective_Pointer + 1) & O_Conclusions(x).Substring(Consequent_Conj1 + 1, O_Conclusions(x).Length -
(Consequent_Conj1 + 2))
                For y = 1 To 2
                    Test1 = 1
                    Candidate_Conclusion = Candidate_Conclusions(y)
                    check_order_of_literals()
                    For z = 1 To Number_of_O_Judgments
                        If Candidate_Conclusion = O_Conclusions(z) Then
                            Test1 = 0
                            If O_LBounds(x) > O_LBounds(z) Then
                                O_LBounds(z) = O_LBounds(x)
                                Inference_Made_Test = 1
                            End If
                        End If
                    Next
                    If Test1 = 1 Then
                        Number_of_O_Judgments = Number_of_O_Judgments + 1
                        O_Conclusions(Number_of_O_Judgments) = Candidate_Conclusion
                        O_LBounds(Number_of_O_Judgments) = O_LBounds(x)
                        Inference_Made_Test = 1
```

```
                End If
            Next
        End If

        If Size_of_Consequent = 2 Then
            Candidate_Conclusions(1) = O_Conclusions(x).Substring(0, Main_Connective_Pointer + 1) & O_Conclusions(x).Substring(Main_Connective_Pointer + 2, Consequent_Conj2 -
(Main_Connective_Pointer + 2))
            Candidate_Conclusions(2) = O_Conclusions(x).Substring(0, Main_Connective_Pointer + 1) & "(" & O_Conclusions(x).Substring(Main_Connective_Pointer + 3,
Consequent_Conj1 - (Main_Connective_Pointer + 3)) & "+" & O_Conclusions(x).Substring(Consequent_Conj2 + 1, O_Conclusions(x).Length - (Consequent_Conj2 + 2)) & ")"
            Candidate_Conclusions(3) = O_Conclusions(x).Substring(0, Main_Connective_Pointer + 1) & "(" & O_Conclusions(x).Substring(Consequent_Conj1 + 1, Consequent_Conj2 -
(Consequent_Conj1 + 2)) & "+" & O_Conclusions(x).Substring(Consequent_Conj2 + 1, O_Conclusions(x).Length - (Consequent_Conj2 + 2)) & ")"
            Candidate_Conclusions(4) = O_Conclusions(x).Substring(0, Main_Connective_Pointer + 1) & O_Conclusions(x).Substring(Main_Connective_Pointer + 3, Consequent_Conj1 -
(Main_Connective_Pointer + 3))
            Candidate_Conclusions(5) = O_Conclusions(x).Substring(0, Main_Connective_Pointer + 1) & O_Conclusions(x).Substring(Consequent_Conj1 + 1, Consequent_Conj2 -
(Consequent_Conj1 + 2))
            Candidate_Conclusions(6) = O_Conclusions(x).Substring(0, Main_Connective_Pointer + 1) & O_Conclusions(x).Substring(Consequent_Conj2 + 1, O_Conclusions(x).Length -
(Consequent_Conj2 + 2))
            For y = 1 To 6
                Test1 = 1
                Candidate_Conclusion = Candidate_Conclusions(y)
                check_order_of_literals()
                For z = 1 To Number_of_O_Judgments
                    If Candidate_Conclusion = O_Conclusions(z) Then
                        Test1 = 0
                        If O_LBounds(x) > O_LBounds(z) Then
                            O_LBounds(z) = O_LBounds(x)
                            Inference_Made_Test = 1
                        End If
                    End If
                Next
                If Test1 = 1 Then
                    Number_of_O_Judgments = Number_of_O_Judgments + 1
                    O_Conclusions(Number_of_O_Judgments) = Candidate_Conclusion
                    O_LBounds(Number_of_O_Judgments) = O_LBounds(x)
                    Inference_Made_Test = 1
                End If
            Next
        End If
    Next
End Sub

Private Sub very_cautious_monotony()
    Dim Test1 As Integer
    Dim Main_Connective_Pointer As Integer
    Dim Pointer As Integer
    Dim Temp_Length As Integer = Number_of_O_Judgments
    Dim Size_of_Consequent As Integer
    Dim Consequent_Conj1 As Integer
    Dim Consequent_Conj2 As Integer
    Dim Candidate_Conclusions(6) As String

    For x = 1 To Temp_Length
        Main_Connective_Pointer = 0
        Test1 = 0
        While Test1 = 0
            If O_Conclusions(x).Chars(Main_Connective_Pointer) = "-" Then Test1 = 1
            Main_Connective_Pointer = Main_Connective_Pointer + 1
        End While
        Main_Connective_Pointer = Main_Connective_Pointer - 1
        Pointer = Main_Connective_Pointer + 1
        Size_of_Consequent = 0
        Test1 = 0
        While Test1 = 0
            If O_Conclusions(x).Chars(Pointer) = "+" Then
                Size_of_Consequent = Size_of_Consequent + 1
                If Size_of_Consequent = 1 Then Consequent_Conj1 = Pointer
                If Size_of_Consequent = 2 Then Consequent_Conj2 = Pointer
            End If
            Pointer = Pointer + 1
            If Pointer = O_Conclusions(x).Length Then Test1 = 1
        End While
        If Size_of_Consequent = 1 Then
            Candidate_Conclusions(1) = "(" & O_Conclusions(x).Substring(0, Main_Connective_Pointer) & "+" & O_Conclusions(x).Substring(Consequent_Conj1 + 1,
O_Conclusions(x).Length - (Consequent_Conj1 + 2)) & ")-" & O_Conclusions(x).Substring(Main_Connective_Pointer + 2, Consequent_Conj1 - (Main_Connective_Pointer + 2))
            Candidate_Conclusions(2) = "(" & O_Conclusions(x).Substring(0, Main_Connective_Pointer) & "+" & O_Conclusions(x).Substring(Main_Connective_Pointer + 2,
Consequent_Conj1 - (Main_Connective_Pointer + 2)) & ")-" & O_Conclusions(x).Substring(Consequent_Conj1 + 1, O_Conclusions(x).Length - (Consequent_Conj1 + 2))
            For y = 1 To 2
                Test1 = 1
                Candidate_Conclusion = Candidate_Conclusions(y)
                check_order_of_literals()
                For z = 1 To Number_of_O_Judgments
                    If Candidate_Conclusion = O_Conclusions(z) Then
                        Test1 = 0
                        If O_LBounds(x) > O_LBounds(z) Then
                            O_LBounds(z) = O_LBounds(x)
                            Inference_Made_Test = 1
                        End If
                    End If
                Next
                If Test1 = 1 Then
                    Number_of_O_Judgments = Number_of_O_Judgments + 1
                    O_Conclusions(Number_of_O_Judgments) = Candidate_Conclusion
                    O_LBounds(Number_of_O_Judgments) = O_LBounds(x)
                    Inference_Made_Test = 1
                End If
            Next
        End If
        If Size_of_Consequent = 2 Then
            Candidate_Conclusions(1) = "(" & O_Conclusions(x).Substring(0, Main_Connective_Pointer) & "+" & O_Conclusions(x).Substring(Consequent_Conj2 + 1,
O_Conclusions(x).Length - (Consequent_Conj2 + 2)) & ")-" & O_Conclusions(x).Substring(Main_Connective_Pointer + 2, Consequent_Conj2 - (Main_Connective_Pointer + 2))
            Candidate_Conclusions(2) = "(" & O_Conclusions(x).Substring(0, Main_Connective_Pointer) & "+" & O_Conclusions(x).Substring(Consequent_Conj1 + 1, Consequent_Conj2 -
(Consequent_Conj1 + 2)) & ")-(" & O_Conclusions(x).Substring(Main_Connective_Pointer + 3, Consequent_Conj1 - (Main_Connective_Pointer + 3)) & "+" &
O_Conclusions(x).Substring(Consequent_Conj2 + 1, O_Conclusions(x).Length - (Consequent_Conj2 + 2)) & ")"
            Candidate_Conclusions(3) = "(" & O_Conclusions(x).Substring(0, Main_Connective_Pointer) & "+" & O_Conclusions(x).Substring(Main_Connective_Pointer + 3,
Consequent_Conj1 - (Main_Connective_Pointer + 3)) & ")-(" & O_Conclusions(x).Substring(Consequent_Conj1 + 1, Consequent_Conj2 - (Consequent_Conj1 + 2)) & "+" &
O_Conclusions(x).Substring(Consequent_Conj2 + 1, O_Conclusions(x).Length - (Consequent_Conj2 + 2)) & ")"
            Candidate_Conclusions(4) = "(" & O_Conclusions(x).Substring(0, Main_Connective_Pointer) & "+(" & O_Conclusions(x).Substring(Consequent_Conj1 + 1, Consequent_Conj2
- (Consequent_Conj1 + 2)) & "+" & O_Conclusions(x).Substring(Consequent_Conj2 + 1, O_Conclusions(x).Length - (Consequent_Conj2 + 2)) & "))-" &
O_Conclusions(x).Substring(Main_Connective_Pointer + 3, Consequent_Conj1 - (Main_Connective_Pointer + 3))
            Candidate_Conclusions(5) = "(" & O_Conclusions(x).Substring(0, Main_Connective_Pointer) & "+(" & O_Conclusions(x).Substring(Main_Connective_Pointer + 3,
Consequent_Conj1 - (Main_Connective_Pointer + 3)) & "+" & O_Conclusions(x).Substring(Consequent_Conj2 + 1, O_Conclusions(x).Length - (Consequent_Conj2 + 2)) & "))-" &
O_Conclusions(x).Substring(Consequent_Conj1 + 1, Consequent_Conj2 - (Consequent_Conj1 + 2))
            Candidate_Conclusions(6) = "(" & O_Conclusions(x).Substring(0, Main_Connective_Pointer) & "+(" & O_Conclusions(x).Substring(Main_Connective_Pointer + 3,
Consequent_Conj1 - (Main_Connective_Pointer + 3)) & "+" & O_Conclusions(x).Substring(Consequent_Conj1 + 1, Consequent_Conj2 - (Consequent_Conj1 + 2)) & "))-" &
O_Conclusions(x).Substring(Consequent_Conj2 + 1, O_Conclusions(x).Length - (Consequent_Conj2 + 2))
            For y = 1 To 6
                Test1 = 1
                Candidate_Conclusion = Candidate_Conclusions(y)
                check_order_of_literals()
                For z = 1 To Number_of_O_Judgments
                    If Candidate_Conclusion = O_Conclusions(z) Then
                        Test1 = 0
                        If O_LBounds(x) > O_LBounds(z) Then
                            O_LBounds(z) = O_LBounds(x)
                            Inference_Made_Test = 1
                        End If
                    End If
```

```
                    Next
                    If Test1 = 1 Then
                        Number_of_O_Judgments = Number_of_O_Judgments + 1
                        O_Conclusions(Number_of_O_Judgments) = Candidate_Conclusion
                        O_LBounds(Number_of_O_Judgments) = O_LBounds(x)
                        Inference_Made_Test = 1
                    End If
                Next
            End If
        Next
    End Sub

    Private Sub exclusive_Or()
        Dim Test1 As Integer
        Dim Main_Connective_Pointer1 As Integer
        Dim Main_Connective_Pointer2 As Integer
        Dim Temp_Length As Integer = Number_of_O_Judgments
        Dim Ante_Conj_Count1 As Integer
        Dim Ante_Conj_Count2 As Integer
        Dim Conj_Pos_One1 As Integer
        Dim Conj_Pos_One2 As Integer
        Dim Conj_Pos_Two1 As Integer
        Dim Conj_Pos_Two2 As Integer
        Dim Alpha1 As String
        Dim Beta1 As String
        Dim Gamma1 As String
        Dim Alpha2 As String
        Dim Beta2 As String
        Dim Gamma2 As String

        For x = 1 To Temp_Length
            For y = x + 1 To Temp_Length
                'If x = 3 And y = 4 Then
                '    x = 3
                'End If

                Main_Connective_Pointer1 = 0
                Main_Connective_Pointer2 = 0
                Ante_Conj_Count1 = 0
                Ante_Conj_Count2 = 0
                Test1 = 0
                While Test1 = 0
                    If O_Conclusions(x).Chars(Main_Connective_Pointer1) = "-" Then Test1 = 1
                    If O_Conclusions(x).Chars(Main_Connective_Pointer1) = "+" Then
                        Ante_Conj_Count1 = Ante_Conj_Count1 + 1
                        If Ante_Conj_Count1 = 1 Then Conj_Pos_One1 = Main_Connective_Pointer1
                        If Ante_Conj_Count1 = 2 Then Conj_Pos_Two1 = Main_Connective_Pointer1
                    End If
                    Main_Connective_Pointer1 = Main_Connective_Pointer1 + 1
                End While
                Main_Connective_Pointer1 = Main_Connective_Pointer1 - 1
                Test1 = 0
                While Test1 = 0
                    If O_Conclusions(y).Chars(Main_Connective_Pointer2) = "-" Then Test1 = 1
                    If O_Conclusions(y).Chars(Main_Connective_Pointer2) = "+" Then
                        Ante_Conj_Count2 = Ante_Conj_Count2 + 1
                        If Ante_Conj_Count2 = 1 Then Conj_Pos_One2 = Main_Connective_Pointer2
                        If Ante_Conj_Count2 = 2 Then Conj_Pos_Two2 = Main_Connective_Pointer2
                    End If
                    Main_Connective_Pointer2 = Main_Connective_Pointer2 + 1
                End While
                Test1 = 0
                Main_Connective_Pointer2 = Main_Connective_Pointer2 - 1
                If Ante_Conj_Count1 > 0 And Ante_Conj_Count1 = Ante_Conj_Count2 Then
                    If Ante_Conj_Count1 = 1 And O_Conclusions(x).Substring(Main_Connective_Pointer1, O_Conclusions(x).Length - Main_Connective_Pointer1) =
O_Conclusions(y).Substring(Main_Connective_Pointer2, O_Conclusions(y).Length - Main_Connective_Pointer2) Then
                        Alpha1 = O_Conclusions(x).Substring(1, Conj_Pos_One1 - 1)
                        Beta1 = O_Conclusions(x).Substring(Conj_Pos_One1 + 1, Main_Connective_Pointer1 - (Conj_Pos_One1 + 2))
                        Alpha2 = O_Conclusions(y).Substring(1, Conj_Pos_One2 - 1)
                        Beta2 = O_Conclusions(y).Substring(Conj_Pos_One2 + 1, Main_Connective_Pointer2 - (Conj_Pos_One2 + 2))
                        If (Alpha1 = Alpha2 And Beta1 = "~" & Beta2) Or (Alpha1 = Alpha2 And "~" & Beta1 = Beta2) Then
                            Candidate_Conclusion = Alpha1 & O_Conclusions(x).Substring(Main_Connective_Pointer1, O_Conclusions(x).Length - Main_Connective_Pointer1)
                            Test1 = 1
                        End If
                        If ("~" & Alpha1 = Alpha2 And Beta1 = Beta2) Or (Alpha1 = "~" & Alpha2 And Beta1 = Beta2) Then
                            Candidate_Conclusion = Beta1 & O_Conclusions(x).Substring(Main_Connective_Pointer1, O_Conclusions(x).Length - Main_Connective_Pointer1)
                            Test1 = 1
                        End If
                        If Test1 = 1 Then
                            For z = 1 To Number_of_O_Judgments
                                If Candidate_Conclusion = O_Conclusions(z) Then
                                    Test1 = 0
                                End If
                            Next
                            If Test1 = 1 Then
                                Number_of_O_Judgments = Number_of_O_Judgments + 1
                                O_Conclusions(Number_of_O_Judgments) = Candidate_Conclusion
                                If O_LBounds(x) > O_LBounds(y) Then
                                    O_LBounds(Number_of_O_Judgments) = O_LBounds(y)
                                Else
                                    O_LBounds(Number_of_O_Judgments) = O_LBounds(x)
                                End If
                                Inference_Made_Test = 1
                            End If
                        End If
                    End If
                    If Ante_Conj_Count1 = 2 And O_Conclusions(x).Substring(Main_Connective_Pointer1, O_Conclusions(x).Length - Main_Connective_Pointer1) =
O_Conclusions(y).Substring(Main_Connective_Pointer2, O_Conclusions(y).Length - Main_Connective_Pointer2) Then
                        Alpha1 = O_Conclusions(x).Substring(2, Conj_Pos_One1 - 2)
                        Beta1 = O_Conclusions(x).Substring(Conj_Pos_One1 + 1, Conj_Pos_Two1 - (Conj_Pos_One1 + 2))
                        Alpha2 = O_Conclusions(y).Substring(2, Conj_Pos_One2 - 2)
                        Beta2 = O_Conclusions(y).Substring(Conj_Pos_One2 + 1, Conj_Pos_Two2 - (Conj_Pos_One2 + 2))
                        Gamma1 = O_Conclusions(x).Substring(Conj_Pos_Two1 + 1, Main_Connective_Pointer1 - (Conj_Pos_Two1 + 2))
                        Gamma2 = O_Conclusions(y).Substring(Conj_Pos_Two2 + 1, Main_Connective_Pointer2 - (Conj_Pos_Two2 + 2))
                        If (Alpha1 = Alpha2 And Beta1 = Beta2 And Gamma1 = "~" & Gamma2) Or (Alpha1 = Alpha2 And Beta1 = Beta2 And "~" & Gamma1 = Gamma2) Then
                            Candidate_Conclusion = "(" & Alpha1 & "+" & Beta1 & ")" & O_Conclusions(x).Substring(Main_Connective_Pointer1, O_Conclusions(x).Length -
Main_Connective_Pointer1)
                            Test1 = 1
                        End If
                        If (Alpha1 = Alpha2 And Beta1 = "~" & Beta2 And Gamma1 = Gamma2) Or (Alpha1 = Alpha2 And "~" & Beta1 = Beta2 And Gamma1 = Gamma2) Then
                            Candidate_Conclusion = "(" & Alpha1 & "+" & Gamma1 & ")" & O_Conclusions(x).Substring(Main_Connective_Pointer1, O_Conclusions(x).Length -
Main_Connective_Pointer1)
                            Test1 = 1
                        End If
                        If (Alpha1 = "~" & Alpha2 And Beta1 = Beta2 And Gamma1 = Gamma2) Or ("~" & Alpha1 = Alpha2 And Beta1 = Beta2 And Gamma1 = Gamma2) Then
                            Candidate_Conclusion = "(" & Beta1 & "+" & Gamma1 & ")" & O_Conclusions(x).Substring(Main_Connective_Pointer1, O_Conclusions(x).Length -
Main_Connective_Pointer1)
                            Test1 = 1
                        End If
                        If Test1 = 1 Then
                            check_order_of_literals()
                            For z = 1 To Number_of_O_Judgments
                                If Candidate_Conclusion = O_Conclusions(z) Then
                                    If O_LBounds(x) > O_LBounds(z) And O_LBounds(y) > O_LBounds(z) Then
                                        If O_LBounds(x) > O_LBounds(y) Then
                                            O_LBounds(z) = O_LBounds(x)
                                        Else
```

```
                                    O_LBounds(z) = O_LBounds(y)
                                End If
                                Inference_Made_Test = 1
                            End If
                            Test1 = 0
                        End If
                    Next
                    If Test1 = 1 Then
                        Number_of_O_Judgments = Number_of_O_Judgments + 1
                        O_Conclusions(Number_of_O_Judgments) = Candidate_Conclusion
                        If O_LBounds(x) > O_LBounds(y) Then
                            O_LBounds(Number_of_O_Judgments) = O_LBounds(y)
                        Else
                            O_LBounds(Number_of_O_Judgments) = O_LBounds(x)
                        End If
                        Inference_Made_Test = 1
                    End If
                End If
            End If
        End If
    Next
Next
End Sub

Private Sub wand()
    Dim Test1 As Integer
    Dim Main_Connective_Pointer1 As Integer
    Dim Main_Connective_Pointer2 As Integer
    Dim Temp_Length As Integer = Number_of_O_Judgments
    Dim ConsPart1 As String
    Dim ConsPart2 As String
    Dim Counter1 As Integer
    Dim Nega As Integer = 0
    Dim Negb As Integer = 0
    Dim Negc As Integer = 0
    Dim Negd As Integer = 0
    Dim A As Integer = 0
    Dim B As Integer = 0
    Dim C As Integer = 0
    Dim D As Integer = 0
    Dim LitCount As Integer

    For x = 1 To Temp_Length
        For y = x + 1 To Temp_Length
            If O_LBounds(x) = 1 Or O_LBounds(y) = 1 Then 'first test for wand applicability

                Main_Connective_Pointer1 = 0
                Main_Connective_Pointer2 = 0
                Test1 = 0
                While Test1 = 0
                    If O_Conclusions(x).Chars(Main_Connective_Pointer1) = "-" Then Test1 = 1
                    Main_Connective_Pointer1 = Main_Connective_Pointer1 + 1
                End While
                Main_Connective_Pointer1 = Main_Connective_Pointer1 - 1
                Test1 = 0
                While Test1 = 0
                    If O_Conclusions(y).Chars(Main_Connective_Pointer2) = "-" Then Test1 = 1
                    Main_Connective_Pointer2 = Main_Connective_Pointer2 + 1
                End While
                Main_Connective_Pointer2 = Main_Connective_Pointer2 - 1

                If O_Conclusions(x).Substring(0, Main_Connective_Pointer1) = O_Conclusions(y).Substring(0, Main_Connective_Pointer2) Then 'second test for wand applicability
                    ConsPart1 = "*" & O_Conclusions(x).Substring(Main_Connective_Pointer1 + 1, O_Conclusions(x).Length - (Main_Connective_Pointer1 + 1)) & "*"
                    ConsPart2 = "*" & O_Conclusions(y).Substring(Main_Connective_Pointer2 + 1, O_Conclusions(y).Length - (Main_Connective_Pointer2 + 1)) & "*"

                    Test1 = 0
                    Counter1 = 1
                    While Test1 = 0
                        If ConsPart1.Chars(Counter1) = "a" Then
                            LitCount = LitCount + 1
                            If ConsPart1.Chars(Counter1 - 1) = "~" Then
                                Nega = 1
                            Else
                                A = 1
                            End If
                        End If
                        If ConsPart1.Chars(Counter1) = "b" Then
                            LitCount = LitCount + 1
                            If ConsPart1.Chars(Counter1 - 1) = "~" Then
                                Negb = 1
                            Else
                                B = 1
                            End If
                        End If
                        If ConsPart1.Chars(Counter1) = "c" Then
                            LitCount = LitCount + 1
                            If ConsPart1.Chars(Counter1 - 1) = "~" Then
                                Negc = 1
                            Else
                                C = 1
                            End If
                        End If
                        If ConsPart1.Chars(Counter1) = "d" Then
                            LitCount = LitCount + 1
                            If ConsPart1.Chars(Counter1 - 1) = "~" Then
                                Negd = 1
                            Else
                                D = 1
                            End If
                        End If
                        If ConsPart1.Chars(Counter1) = "*" Then Test1 = 1
                        Counter1 = Counter1 + 1
                    End While

                    Test1 = 0
                    Counter1 = 1
                    While Test1 = 0
                        If ConsPart2.Chars(Counter1) = "a" Then
                            LitCount = LitCount + 1
                            If ConsPart2.Chars(Counter1 - 1) = "~" Then
                                Nega = 1
                            Else
                                A = 1
                            End If
                        End If
                        If ConsPart2.Chars(Counter1) = "b" Then
                            LitCount = LitCount + 1
                            If ConsPart2.Chars(Counter1 - 1) = "~" Then
                                Negb = 1
                            Else
                                B = 1
                            End If
                        End If
                        If ConsPart2.Chars(Counter1) = "c" Then
                            LitCount = LitCount + 1
                            If ConsPart2.Chars(Counter1 - 1) = "~" Then
                                Negc = 1
```

```
        Else
            C = 1
        End If
    End If
    If ConsPart2.Chars(Counter1) = "d" Then
        LitCount = LitCount + 1
        If ConsPart2.Chars(Counter1 - 1) = "~" Then
            Negd = 1
        Else
            D = 1
        End If
    End If
    If ConsPart2.Chars(Counter1) = "*" Then Test1 = 1
    Counter1 = Counter1 + 1
End While

If Nega = 1 And A = 1 Then
    Nega = 0
    A = 0
    LitCount = LitCount - 2
End If
If Negb = 1 And B = 1 Then
    Negb = 0
    B = 0
    LitCount = LitCount - 2
End If
If Negc = 1 And C = 1 Then
    Negc = 0
    B = 0
    LitCount = LitCount - 2
End If
If Negd = 1 And D = 1 Then
    Negd = 0
    D = 0
    LitCount = LitCount - 2
End If

If LitCount > 0 Then
    Select Case LitCount
        Case 1
            If A = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-a"
            If Nega = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-~a"
            If B = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-b"
            If Negb = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-~b"
            If C = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-c"
            If Negc = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-~c"
            If D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-d"
            If Negd = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-~d"
        Case 2
            If (A = 1 Or Nega = 1) And (B = 1 Or Negb = 1) Then
                If A = 1 And B = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(a+b)"
                If A = 1 And B = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(a+~b)"
                If A = 0 And B = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~a+b)"
                If A = 0 And B = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~a+~b)"
            End If
            If (A = 1 Or Nega = 1) And (C = 1 Or Negc = 1) Then
                If A = 1 And C = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(a+c)"
                If A = 1 And C = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(a+~c)"
                If A = 0 And C = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~a+c)"
                If A = 0 And C = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~a+~c)"
            End If
            If (A = 1 Or Nega = 1) And (D = 1 Or Negd = 1) Then
                If A = 1 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(a+d)"
                If A = 1 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(a+~d)"
                If A = 0 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~a+d)"
                If A = 0 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~a+~d)"
            End If
            If (B = 1 Or Negb = 1) And (C = 1 Or Negc = 1) Then
                If B = 1 And C = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(b+c)"
                If B = 1 And C = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(b+~c)"
                If B = 0 And C = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~b+c)"
                If B = 0 And C = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~b+~c)"
            End If
            If (B = 1 Or Negb = 1) And (D = 1 Or Negd = 1) Then
                If B = 1 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(b+d)"
                If B = 1 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(b+~d)"
                If B = 0 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~b+d)"
                If B = 0 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~b+~d)"
            End If
            If (C = 1 Or Negc = 1) And (D = 1 Or Negd = 1) Then
                If C = 1 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(b+d)"
                If C = 1 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(b+~d)"
                If C = 0 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~b+d)"
                If C = 0 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-(~b+~d)"
            End If
        Case 3
            If Not (A = 1 And Nega = 1) Then
                If B = 1 And C = 1 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((b+c)+d)"
                If B = 1 And C = 1 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((b+c)+~d)"
                If B = 1 And C = 0 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((b+~c)+d)"
                If B = 1 And C = 0 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((b+~c)+~d)"
                If B = 0 And C = 1 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~b+c)+d)"
                If B = 0 And C = 1 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~b+c)+~d)"
                If B = 0 And C = 0 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~b+~c)+d)"
                If B = 0 And C = 0 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~b+~c)+~d)"
            End If
            If Not (B = 1 And Negb = 1) Then
                If A = 1 And C = 1 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+c)+d)"
                If A = 1 And C = 1 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+c)+~d)"
                If A = 1 And C = 0 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+~c)+d)"
                If A = 1 And C = 0 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+~c)+~d)"
                If A = 0 And C = 1 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+c)+d)"
                If A = 0 And C = 1 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+c)+~d)"
                If A = 0 And C = 0 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+~c)+d)"
                If A = 0 And C = 0 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+~c)+~d)"
            End If
            If Not (C = 1 And Negc = 1) Then
                If A = 1 And B = 1 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+b)+d)"
                If A = 1 And B = 1 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+b)+~d)"
                If A = 1 And B = 0 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+~b)+d)"
                If A = 1 And B = 0 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+~b)+~d)"
                If A = 0 And B = 1 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+b)+d)"
                If A = 0 And B = 1 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+b)+~d)"
                If A = 0 And B = 0 And D = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+~b)+d)"
                If A = 0 And B = 0 And D = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+~b)+~d)"
            End If
            If Not (D = 1 And Negd = 1) Then
                If A = 1 And B = 1 And C = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+b)+c)"
                If A = 1 And B = 1 And C = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+b)+~c)"
                If A = 1 And B = 0 And C = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+~b)+c)"
                If A = 1 And B = 0 And C = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((a+~b)+~c)"
                If A = 0 And B = 1 And C = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+b)+c)"
                If A = 0 And B = 1 And C = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+b)+~c)"
                If A = 0 And B = 0 And C = 1 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+~b)+c)"
                If A = 0 And B = 0 And C = 0 Then Candidate_Conclusion = O_Conclusions(x).Substring(0, Main_Connective_Pointer1) & "-((~a+~b)+~c)"
            End If
    End Select
```

```
                        Test1 = 1
                        For z = 1 To Number_of_O_Judgments
                            If Candidate_Conclusion = O_Conclusions(z) Then
                                If O_LBounds(x) > O_LBounds(z) And O_LBounds(y) > O_LBounds(z) Then
                                    If O_LBounds(x) > O_LBounds(y) Then
                                        O_LBounds(z) = O_LBounds(x)
                                    Else
                                        O_LBounds(z) = O_LBounds(y)
                                    End If
                                    Inference_Made_Test = 1
                                End If
                                Test1 = 0
                            End If
                        Next
                        If Test1 = 1 Then
                            Number_of_O_Judgments = Number_of_O_Judgments + 1
                            O_Conclusions(Number_of_O_Judgments) = Candidate_Conclusion
                            If O_LBounds(x) > O_LBounds(y) Then
                                O_LBounds(Number_of_O_Judgments) = O_LBounds(y)
                            Else
                                O_LBounds(Number_of_O_Judgments) = O_LBounds(x)
                            End If
                            Inference_Made_Test = 1
                        End If

                    End If

                End If

            End If
        Next
    Next

End Sub

Private Sub check_order_of_literals()
    Dim Test1 As Integer
    Dim Counter1 As Integer
    Dim Holder As Integer
    Dim String_Holder As String
    Dim Subject_Conditional As String
    Dim Ante_Literal_Holder(3) As String
    Dim Cons_Literal_Holder(3) As String
    Dim Literal_Holder_Counter As Integer
    Dim Literal_Counter As Integer
    Dim Cons_Now As Integer
    Dim Ante_Length As Integer
    Dim Cons_Length As Integer
    Dim apos As Integer
    Dim bpos As Integer
    Dim cpos As Integer
    Dim dpos As Integer
    Dim Neg As Integer

    Subject_Conditional = "*" & Candidate_Conclusion & "*"

    apos = 0
    bpos = 0
    cpos = 0
    dpos = 0
    Cons_Now = 0
    Literal_Holder_Counter = 0
    Literal_Counter = 0
    Counter1 = 1
    Test1 = 0

    While Test1 = 0
        Neg = 0
        If Subject_Conditional.Chars(Counter1) = "-" Then
            Cons_Now = 1
            Ante_Length = Literal_Holder_Counter
            Literal_Holder_Counter = 0
        End If
        If Subject_Conditional.Chars(Counter1) = "a" Then
            If Subject_Conditional.Chars(Counter1 - 1) = "~" Then Neg = 1
            Literal_Holder_Counter = Literal_Holder_Counter + 1
            Literal_Counter = Literal_Counter + 1
            apos = Literal_Counter
            If Cons_Now = 0 Then
                If Neg = 1 Then
                    Ante_Literal_Holder(Literal_Holder_Counter) = "~a"
                Else
                    Ante_Literal_Holder(Literal_Holder_Counter) = "a"
                End If
            Else
                If Neg = 1 Then
                    Cons_Literal_Holder(Literal_Holder_Counter) = "~a"
                Else
                    Cons_Literal_Holder(Literal_Holder_Counter) = "a"
                End If
            End If
        End If
        If Subject_Conditional.Chars(Counter1) = "b" Then
            If Subject_Conditional.Chars(Counter1 - 1) = "~" Then Neg = 1
            Literal_Holder_Counter = Literal_Holder_Counter + 1
            Literal_Counter = Literal_Counter + 1
            bpos = Literal_Counter
            If Cons_Now = 0 Then
                If Neg = 1 Then
                    Ante_Literal_Holder(Literal_Holder_Counter) = "~b"
                Else
                    Ante_Literal_Holder(Literal_Holder_Counter) = "b"
                End If
            Else
                If Neg = 1 Then
                    Cons_Literal_Holder(Literal_Holder_Counter) = "~b"
                Else
                    Cons_Literal_Holder(Literal_Holder_Counter) = "b"
                End If
            End If
        End If
        If Subject_Conditional.Chars(Counter1) = "c" Then
            If Subject_Conditional.Chars(Counter1 - 1) = "~" Then Neg = 1
            Literal_Holder_Counter = Literal_Holder_Counter + 1
            Literal_Counter = Literal_Counter + 1
            cpos = Literal_Counter
            If Cons_Now = 0 Then
                If Neg = 1 Then
                    Ante_Literal_Holder(Literal_Holder_Counter) = "~c"
                Else
                    Ante_Literal_Holder(Literal_Holder_Counter) = "c"
                End If
            Else
                If Neg = 1 Then
                    Cons_Literal_Holder(Literal_Holder_Counter) = "~c"
                Else
```

```
                Cons_Literal_Holder(Literal_Holder_Counter) = "c"
            End If
        End If
    End If
    If Subject_Conditional.Chars(Counter1) = "d" Then
        If Subject_Conditional.Chars(Counter1 - 1) = "~" Then Neg = 1
        Literal_Holder_Counter = Literal_Holder_Counter + 1
        Literal_Counter = Literal_Counter + 1
        dpos = Literal_Counter
        If Cons_Now = 0 Then
            If Neg = 1 Then
                Ante_Literal_Holder(Literal_Holder_Counter) = "~d"
            Else
                Ante_Literal_Holder(Literal_Holder_Counter) = "d"
            End If
        Else
            If Neg = 1 Then
                Cons_Literal_Holder(Literal_Holder_Counter) = "~d"
            Else
                Cons_Literal_Holder(Literal_Holder_Counter) = "d"
            End If
        End If
    End If
    If Subject_Conditional.Chars(Counter1 + 1) = "*" Then
        Test1 = 1
    End If
    Counter1 = Counter1 + 1
End While
Cons_Length = Literal_Holder_Counter
Test1 = 0
While Test1 = 0
    Test1 = 1
    If apos > bpos And Not bpos = 0 And apos <= Ante_Length And bpos <= Ante_Length Then
        String_Holder = Ante_Literal_Holder(bpos)
        Ante_Literal_Holder(bpos) = Ante_Literal_Holder(apos)
        Ante_Literal_Holder(apos) = String_Holder
        Holder = bpos
        bpos = apos
        apos = Holder
        Test1 = 0
    End If
    If apos > bpos And Not bpos = 0 And apos > Ante_Length And bpos > Ante_Length Then
        String_Holder = Ante_Literal_Holder(bpos)
        Cons_Literal_Holder(bpos - Ante_Length) = Cons_Literal_Holder(apos - Ante_Length)
        Cons_Literal_Holder(apos - Ante_Length) = String_Holder
        Holder = bpos
        bpos = apos
        apos = Holder
        Test1 = 0
    End If
    If apos > cpos And Not cpos = 0 And apos <= Ante_Length And cpos <= Ante_Length Then
        String_Holder = Ante_Literal_Holder(cpos)
        Ante_Literal_Holder(cpos) = Ante_Literal_Holder(apos)
        Ante_Literal_Holder(apos) = String_Holder
        Holder = cpos
        cpos = apos
        apos = Holder
        Test1 = 0
    End If
    If apos > cpos And Not cpos = 0 And apos > Ante_Length And cpos > Ante_Length Then
        String_Holder = Ante_Literal_Holder(cpos)
        Cons_Literal_Holder(cpos - Ante_Length) = Cons_Literal_Holder(apos - Ante_Length)
        Cons_Literal_Holder(apos - Ante_Length) = String_Holder
        Holder = cpos
        cpos = apos
        apos = Holder
        Test1 = 0
    End If
    If apos > dpos And Not dpos = 0 And apos <= Ante_Length And dpos <= Ante_Length Then
        String_Holder = Ante_Literal_Holder(dpos)
        Ante_Literal_Holder(dpos) = Ante_Literal_Holder(apos)
        Ante_Literal_Holder(apos) = String_Holder
        Holder = dpos
        dpos = apos
        apos = Holder
        Test1 = 0
    End If
    If apos > dpos And Not dpos = 0 And apos > Ante_Length And dpos > Ante_Length Then
        String_Holder = Ante_Literal_Holder(dpos)
        Cons_Literal_Holder(dpos - Ante_Length) = Cons_Literal_Holder(apos - Ante_Length)
        Cons_Literal_Holder(apos - Ante_Length) = String_Holder
        Holder = dpos
        dpos = apos
        apos = Holder
        Test1 = 0
    End If
    If bpos > cpos And Not cpos = 0 And bpos <= Ante_Length And cpos <= Ante_Length Then
        String_Holder = Ante_Literal_Holder(cpos)
        Ante_Literal_Holder(cpos) = Ante_Literal_Holder(bpos)
        Ante_Literal_Holder(bpos) = String_Holder
        Holder = cpos
        cpos = bpos
        bpos = Holder
        Test1 = 0
    End If
    If bpos > cpos And Not cpos = 0 And bpos > Ante_Length And cpos > Ante_Length Then
        String_Holder = Ante_Literal_Holder(cpos)
        Cons_Literal_Holder(cpos - Ante_Length) = Cons_Literal_Holder(bpos - Ante_Length)
        Cons_Literal_Holder(bpos - Ante_Length) = String_Holder
        Holder = cpos
        cpos = bpos
        bpos = Holder
        Test1 = 0
    End If
    If bpos > dpos And Not dpos = 0 And bpos <= Ante_Length And dpos <= Ante_Length Then
        String_Holder = Ante_Literal_Holder(dpos)
        Ante_Literal_Holder(dpos) = Ante_Literal_Holder(bpos)
        Ante_Literal_Holder(bpos) = String_Holder
        Holder = dpos
        dpos = bpos
        bpos = Holder
        Test1 = 0
    End If
    If bpos > dpos And Not dpos = 0 And bpos > Ante_Length And dpos > Ante_Length Then
        String_Holder = Ante_Literal_Holder(dpos)
        Cons_Literal_Holder(dpos - Ante_Length) = Cons_Literal_Holder(bpos - Ante_Length)
        Cons_Literal_Holder(bpos - Ante_Length) = String_Holder
        Holder = dpos
        dpos = bpos
        bpos = Holder
        Test1 = 0
    End If
    If cpos > dpos And Not dpos = 0 And cpos <= Ante_Length And dpos <= Ante_Length Then
        String_Holder = Ante_Literal_Holder(dpos)
        Ante_Literal_Holder(dpos) = Ante_Literal_Holder(cpos)
        Ante_Literal_Holder(cpos) = String_Holder
        Holder = dpos
```

```
            dpos = cpos
            cpos = Holder
            Test1 = 0
        End If
        If cpos > dpos And Not dpos = 0 And cpos > Ante_Length And dpos > Ante_Length Then
            String_Holder = Ante_Literal_Holder(dpos)
            Cons_Literal_Holder(dpos - Ante_Length) = Cons_Literal_Holder(cpos - Ante_Length)
            Cons_Literal_Holder(cpos - Ante_Length) = String_Holder
            Holder = dpos
            dpos = cpos
            cpos = Holder
            Test1 = 0
        End If
    End While
    If Ante_Length = 1 And Cons_Length = 2 Then
        Candidate_Conclusion = Ante_Literal_Holder(1) & "-(" & Cons_Literal_Holder(1) & "+" & Cons_Literal_Holder(2) & ")"
    End If
    If Ante_Length = 1 And Cons_Length = 3 Then
        Candidate_Conclusion = Ante_Literal_Holder(1) & "-((" & Cons_Literal_Holder(1) & "+" & Cons_Literal_Holder(2) & ")+" & Cons_Literal_Holder(3) & ")"
    End If
    If Ante_Length = 2 And Cons_Length = 1 Then
        Candidate_Conclusion = "(" & Ante_Literal_Holder(1) & "+" & Ante_Literal_Holder(2) & ")-" & Cons_Literal_Holder(1)
    End If
    If Ante_Length = 2 And Cons_Length = 2 Then
        Candidate_Conclusion = "(" & Ante_Literal_Holder(1) & "+" & Ante_Literal_Holder(2) & ")-(" & Cons_Literal_Holder(1) & Cons_Literal_Holder(2) & ")"
    End If
    If Ante_Length = 3 And Cons_Length = 1 Then
        Candidate_Conclusion = "((" & Ante_Literal_Holder(1) & "+" & Ante_Literal_Holder(2) & ")+" & Ante_Literal_Holder(3) & ")-" & Cons_Literal_Holder(1)
    End If

End Sub

Private Sub print_stuff()
    Dim FILE_NAME As String = "C:\data1.txt"
    Dim objWriter As New System.IO.StreamWriter(FILE_NAME)

    objWriter.WriteLine("Base_Conditionals:")
    For x = 1 To Number_of_Base_Conditionals
        objWriter.WriteLine(Clue_Propositions(x))
    Next
    objWriter.WriteLine("")

    objWriter.WriteLine("O_Conclusions:")
    For y = 1 To Number_of_Props_Queried
        For x = 1 To Number_of_O_Judgments
            If O_Conclusions(x) = Queried_Propositions(y) Then
                objWriter.WriteLine("")
                objWriter.WriteLine(O_Conclusions(x))
                objWriter.WriteLine(O_LBounds(x))
                objWriter.WriteLine(Queried_Probabilities(y))
            End If
        Next
    Next
    objWriter.WriteLine("")

    objWriter.WriteLine("P_Conclusions:")
    For x = 1 To Number_of_P_Judgments
        For y = 1 To Number_of_Props_Queried
            If P_Conclusions(x) = Queried_Propositions(y) Then
                objWriter.WriteLine("")
                objWriter.WriteLine(P_Conclusions(x))
                objWriter.WriteLine(P_LBounds(y))
                objWriter.WriteLine(Queried_Probabilities(y))
            End If
        Next
    Next
    objWriter.WriteLine("")

    objWriter.WriteLine("Z_Conclusions:")
    For x = 1 To Number_of_Z_Judgments
        For y = 1 To Number_of_Props_Queried
            If Z_Conclusions(x) = Queried_Propositions(y) Then
                objWriter.WriteLine("")
                objWriter.WriteLine(Z_Conclusions(x))
                objWriter.WriteLine(Z_LBounds(y))
                objWriter.WriteLine(Queried_Probabilities(y))
            End If
        Next
    Next
    objWriter.WriteLine("")

    'objWriter.WriteLine("Z*_Conclusions:")
    'For x = 1 To Number_of_Zs_Judgments
    '    For y = 1 To Number_of_Props_Queried
    '        If Z_Conclusions(x) = Queried_Propositions(y) Then
    '            objWriter.WriteLine("")
    '            objWriter.WriteLine(Zs_Conclusions(x))
    '            objWriter.WriteLine(Zs_LBounds(y))
    '            objWriter.WriteLine(Queried_Probabilities(y))
    '        End If
    '    Next
    'Next
    'objWriter.WriteLine("")

    objWriter.WriteLine("Classical_Conclusions:")
    For x = 1 To Number_of_Classical_Judgments
        For y = 1 To Number_of_Props_Queried
            If Classical_Conclusions(x) = Queried_Propositions(y) Then
                objWriter.WriteLine("")
                objWriter.WriteLine(Classical_Conclusions(x))
                objWriter.WriteLine(Classical_LBounds(y))
                objWriter.WriteLine(Queried_Probabilities(y))
            End If
        Next
    Next
    objWriter.WriteLine("")

    objWriter.Close()
End Sub

Private Sub print_stuff2()
    Dim FILE_NAME As String = "C:\data2.txt"
    Dim objWriter As New System.IO.StreamWriter(FILE_NAME)

    objWriter.WriteLine("Base_Conditionals:")
    For x = 1 To Number_of_Base_Conditionals
        objWriter.WriteLine(Clue_Propositions(x))
    Next
    objWriter.WriteLine("")

    objWriter.WriteLine("Number_of_Runs:")
    objWriter.WriteLine(Number_of_Runs)
    objWriter.WriteLine("")

    'objWriter.WriteLine("High_Value:")
    'objWriter.WriteLine(High_Value)
    'objWriter.WriteLine("")
```

```
        'objWriter.WriteLine("Low_Value:")
        'objWriter.WriteLine(Low_Value)
        'objWriter.WriteLine("")

        objWriter.WriteLine("Minimum_Probability_for_Base_Conditionals:")
        objWriter.WriteLine(Min_Prob_for_Base_Conds)
        objWriter.WriteLine("")

        'objWriter.WriteLine("Degree_of_Normicity:")
        'objWriter.WriteLine(Degree_of_Normicity)
        'objWriter.WriteLine("")

        objWriter.WriteLine("")
        objWriter.WriteLine("Number_of_O_Judgments:")
        objWriter.WriteLine(Number_of_O_Judgments)
        objWriter.WriteLine("Number_of_P_Judgments:")
        objWriter.WriteLine(Number_of_P_Judgments)
        objWriter.WriteLine("Number_of_Z_Judgments:")
        objWriter.WriteLine(Number_of_Z_Judgments)
        'objWriter.WriteLine("Number_of_Z*_Judgments:")
        'objWriter.WriteLine(Number_of_Zs_Judgments)
        objWriter.WriteLine("Number_of_Classical_Judgments:")
        objWriter.WriteLine(Number_of_Classical_Judgments)

        objWriter.WriteLine("")
        objWriter.WriteLine("O_Errors:")
        objWriter.WriteLine(O_Errors)
        objWriter.WriteLine("P_Errors:")
        objWriter.WriteLine(P_Errors)
        objWriter.WriteLine("Z_Errors:")
        objWriter.WriteLine(Z_Errors)
        'objWriter.WriteLine("Z*_Errors:")
        'objWriter.WriteLine(Zs_Errors)
        objWriter.WriteLine("Classical_Errors:")
        objWriter.WriteLine(Classical_Errors)

        objWriter.WriteLine("")
        objWriter.WriteLine("Total O_AvG_Score:")
        objWriter.WriteLine(O_Score)
        objWriter.WriteLine("Total P_AvG_Score:")
        objWriter.WriteLine(P_Score)
        objWriter.WriteLine("Total Z_AvG_Score:")
        objWriter.WriteLine(Z_Score)
        'objWriter.WriteLine("Total Z*_Score:")
        'objWriter.WriteLine(Zs_Score)
        objWriter.WriteLine("Total Classical_AvG_Score:")
        objWriter.WriteLine(Classical_Score)

        objWriter.WriteLine("")
        'objWriter.WriteLine("Punishment_Constant:")
        'objWriter.WriteLine(Punishment_Constant)
        objWriter.WriteLine("Total O_PIR_Score:")
        objWriter.WriteLine(PIR_O_Score)
        objWriter.WriteLine("Total P_PIR_Score:")
        objWriter.WriteLine(PIR_P_Score)
        objWriter.WriteLine("Total Z_PIR_Score:")
        objWriter.WriteLine(PIR_Z_Score)
        'objWriter.WriteLine("PIR_Z*_Score:")
        'objWriter.WriteLine(PIR_Zs_Score)
        objWriter.WriteLine("Total Classical_PIR_Score:")
        objWriter.WriteLine(PIR_Classical_Score)

        objWriter.WriteLine("")
        'objWriter.WriteLine("SPunishment_Constant:")
        'objWriter.WriteLine(SPunishment_Constant)
        'objWriter.WriteLine("Total O_SPIR_Score:")
        'objWriter.WriteLine(SPIR_O_Score)
        'objWriter.WriteLine("Total P_SPIR_Score:")
        'objWriter.WriteLine(SPIR_P_Score)
        objWriter.WriteLine("Total Z_SPIR_Score:")
        objWriter.WriteLine(SPIR_Z_Score)
        'objWriter.WriteLine("SPIR_Z*_Score:")
        'objWriter.WriteLine(SPIR_Zs_Score)
        objWriter.WriteLine("Total Classical_SPIR_Score:")
        objWriter.WriteLine(SPIR_Classical_Score)

        objWriter.Close()
    End Sub

End Class
```