Pathfinder: A Windows Console Application to Help Draw Finite State Machines Using Visio and Find the Paths in Excel

James Cornett
Student, Grand Valley State University

Follow this and additional works at: http://scholarworks.gvsu.edu/cistechlib

Recommended Citation
http://scholarworks.gvsu.edu/cistechlib/215

This Project is brought to you for free and open access by the School of Computing and Information Systems at ScholarWorks@GVSU. It has been accepted for inclusion in Technical Library by an authorized administrator of ScholarWorks@GVSU. For more information, please contact scholarworks@gvsu.edu.
Pathfinder: A Windows Console Application to Help Draw Finite State Machines Using Visio and Find the Paths in Excel

Jim Cornett, Paul Jorgensen
School of Computing and Information Systems, Grand Valley State University, Allendale, USA
Email: cornett@mail.gvsu.edu, jorgensp@gvsu.edu

Abstract
Tools to perform finite state machine path analysis for students are typically standalone applications. They offer little documentation and are not intuitive. Pathfinder is a Windows console application that allows the use of Microsoft Visio and Excel products to be used for drawing and determining the paths in a finite state machine. The use of Microsoft Office tools allows the user to the ability to perform the task of analysis using applications with which most are familiar.

1. Introduction
Finite state machine analysis is a common if not mandatory part of the curriculum of any software engineering program. The modeling tools available to the student in these scholastic programs are generally standalone applications. The applications typically provide rudimentary drawing and data entry capabilities. Pathfinder allows the student to use common Microsoft Office tools in order to develop finite state machine drawings and perform path analysis.

2. Pathfinder
Pathfinder was developed as a tool for students to more easily draw and analyze finite state machines. It does this by the student placing the state and event information in a Microsoft Excel spreadsheet inside an Excel workbook. Pathfinder takes the data and creates a Microsoft Visio drawing of the data. Pathfinder then returns to Excel and opens another spreadsheet inside the workbook. On this spreadsheet, Pathfinder takes initial state and final state information from the previous spreadsheet and performs a path analysis. The path analysis determines all of the paths through the state machine, counts and records them on the newly created spreadsheet. At this point, the user is able to analyze results of the path analysis. The user is also able to manipulate the Visio drawing as they see fit to create the graphical state machine representation.

2.1. Implementation
Pathfinder is a standard Windows console application implemented using Microsoft Visual Studio and the Microsoft VB.net Framework. The design can be broken into two separate parts. First is the use of the user data in Excel spreadsheet to modify the Visio template and create a new Excel worksheet for the path analysis. The second portion of the program is a recursive call to find paths from a given initial and final state.

When Pathfinder is launched, the flow of a program is as follows:

- The user defined Excel workbook is opened.
- The Visio template is opened and saved as a new Visio drawing file.
- The state and transition information is transferred from one sheet to a new sheet in the Excel workbook.
- The user defined state information is used rename the state shapes in the Visio document.
- The user defined “from state” and “to state” information is used to create connections between states.
- The connections are labeled with event and action information
- The state and event information is transferred from one worksheet to a new worksheet in the Excel workbook.
- The initial state and final state for the path analysis are retrieved from the worksheet and placed in a recursive function.
- The recursive function places states as they are found into the spreadsheet for the current path.
- The recursive function starts at the initial state and traverses the state machine until either the final state is found or there are no other next states for the current state and the unused state / event queue is empty.
- After all the paths are found, the program exits leaving the user to view the paths in the Excel workbook and manipulate the Visio drawing as they see fit.

2.2. Microsoft Visio Drawing and New Worksheet Creation

The creation of the Visio drawing is fairly straightforward. The Visual Basic code required to perform the operation is listed in Appendix I. Most of the work is performed using Microsoft’s Office interop functionality for both Excel and Visio as can be seen in the code. To illustrate the operation an example using a garage door opener will be presented.

The first step is for the user to place the state and transition information in the Excel worksheet as shown in Figure 1.

![Figure 1 Garage Door Opener Finite State Machine Information in Excel](image)

There are three sets of information provided in the worksheet. The first is the state, event and action names. The second is the transitions “from” and “to” states with causes and outputs. Third is the initial and final state for the path analysis. The user saves the worksheet in their documents folder.
When pathfinder is launched it opens the Visio template shown in Figure 2.

Pathfinder then matches the state names with data from the Excel worksheet and renames the state circles to include the state names. Pathfinder then deletes all unused stats and begins placing connectors to represent the state transitions. Lastly, the connectors are labeled with event and action numbers. The result is shown in Figure 3.
This allows the user to easily manipulate the drawing using Visio. The garage door example is shown in Figure 4 after some simple manipulations.

![Figure 4 Visio Drawing of Garage Door Finite State Machine after Modifications in Visio](image)

The initial part of the program also creates a second worksheet to perform the path analysis. The transition data was copied to this spreadsheet in a used area to ease the amount of coding required for the recursive call.

### 2.3 Path Analysis
The path analysis is performed using a recursive function. The Visual Basic code for the function can be found in Appendix II. The flowchart for the pathfinding function is shown in Figure 5. The pathfinding function keeps a queue of the used and unused state / event combinations. The queues are text lists and the individual entries contain three pieces of information, the state, the number of times the state has been entered and the event used to exit the state. The number of times the state has been entered is stored to account for loops in the state machine. In the garage door example when traversing from s01 to s02, the first time in s05 the unused queue would record s051e02, s051e04, s051e05 with 1 being the number of times in s05. The used queue would record s051e01. The second time though s05 event e01 is disregarded and s052e02 is placed in the used queue. The unused queue would then add s052e04 and s052e05 to the end of the unused queue list. The ability to account for loops adds considerable complexity to the program.

The unused queue is used to determine how far to back up in the event of either finding a state with no unused events or in the case of entering the last state. If no unused event is found for a state the program will back up to the last unused state / event and continue searching for the final state. If the final state is reached the unused event queue is used to determine where to start the next path without going back to the initial state.

As can be seen in the flow chart there are three paths to determine that the current state has events available to be used. Once a state / event is chosen the next state is determined using the transition information. The next state is then passed back to the recursive function along with the used and unused state / event queues.
Begin Recursive

Is last state?

Add state to path

Add state to path

Find events for current state

Unused state/events?

Start new path from initial state to last unused state/event

Unused state/event queue empty?

Return to last queued unused state/event

Add state/event to used queue

Choose next state

Call Recursive

Return null

Unused events for current state?

Unused state/event queue empty?

Return null

Figure 5  Pathfinder Recursive Function Flow Chart
The output of the pathfinding for the garage door example can be seen in the Figure 6.

![Figure 6 Output of Pathfinder for the Garage Door Opener](image)

The paths from s01 to s04 are shown in the worksheet along with a count of the total number of paths. These initial and final states were chosen because they best demonstrated the path and loop traversing of the Pathfinder program. It should be noted what looks to be duplicate paths is actually two different paths created by different events leading from state s05 to state s06.

### 3. Conclusion

In conclusion, Pathfinder provides an automated process to help students draw and find paths for finite state machines. Though there are other state machine analysis applications that are available to the student, none have the functionalities of the Microsoft Office products. These products are typically either free to students through the Microsoft Dreamspark program or available at reduced cost though Microsoft student purchase programs.
Appendix I

Imports Microsoft.Office.Interop.Excel
Imports Microsoft.Office.Interop
Imports Microsoft.Office.Interop.Visio

Module Module1

' Dim NewBook As Workbook
' Dim returnValue As Object

Sub Main()
    Dim MyExcel = New Excel.Application
' MyExcel.Workbooks.Add()
    MyExcel.Visible = True

    Dim xlWorkBook As ExcelWorkbook
    xlWorkBook = MyExcel.Workbooks.Open(xlDocPath)

    ' ''Save template as workbook
    ' xlWorkBook.SaveAs(xlDocPathSave)

    Dim vsoApp = New Visio.Application
    vsoApp.Visible = True
    Dim vsoDocument As Visio.Document
'Delete Shapes
Dim vsoPage As Visio.Page
Dim shps As Visio.Shapes
Dim shp As Visio.Shape
vsoPage = vsoApp.ActivePage
shps = vsoPage.Shapes

'Change text and delete unused shapes
Dim x As Integer
For x = 1 To 16
    Dim cellText As String
    Dim stateText As String
    Dim conText As String
    If String.IsNullOrEmpty(MyExcel.Cells((x + 1), 2).Value) Then
        stateText = MyExcel.Cells((x + 1), 1).Text
        shp = shps.Item(stateText)
        shp.Delete()
    Else
        stateText = MyExcel.Cells((x + 1), 1).Text
        cellText = MyExcel.Cells((x + 1), 2).Text
        conText = stateText & "" & cellText
        shp = shps.Item(stateText)
        shp.Text() = conText
    End If
Next

Dim y As Integer
Dim shapeUsedQueue As New List(Of Integer)

For y = 1 To 22
    Dim vsoShape1 As Visio.Shape
    Dim vsoShape2 As Visio.Shape
    Dim newConnector As Visio.Shape
    Dim vsoShape1Text As String
    Dim vsoShape2Text As String
    Dim vsoConTextEvent As String
    Dim vsoConTextAction As String
    Dim vsoConCon As String
If String.IsNullOrEmpty(MyExcel.Cells((y + 1), 6).Value) Then
    Exit For
Else

    vsoShape1Text = MyExcel.Cells((y + 1), 6).Text
    vsoShape2Text = MyExcel.Cells((y + 1), 7).Text
    vsoShape1 = shps.Item(vsoShape1Text)
    vsoShape2 = shps.Item(vsoShape2Text)
    vsoShape1.AutoConnect(vsoShape2, VisAutoConnectDir.visAutoConnectDirNone)

    For Each c1 In vsoShape1.FromConnects
        For Each c2 In c1.FromSheet.Connects
            If ((c2.ToSheet.ID = vsoShape2.ID)) Then
                newConnector = c2.FromSheet
                If Not (shapeUsedQueue.Contains(newConnector.ID)) Then
                    shapeUsedQueue.Add(newConnector.ID)
                    vsoConTextEvent = MyExcel.Cells((y + 1), 8).Text
                    vsoConTextAction = MyExcel.Cells((y + 1), 9).Text
                    vsoConCon = vsoConTextEvent & "/" & vsoConTextAction
                    newConnector.Text() = vsoConCon
                    newConnector.CellsU("EndArrow").Formula = "4"
                    newConnector.CellsU("ConLineRouteExt").Formula = "2"
                    newConnector.CellsU("ShapeRouteStyle").Formula = "1"
                End If
            End If
        Next
    Next
End If

vsoDocument.SaveAs(vsoDocPathSave)

'Start new worksheet for path analysis
Dim xlsSheet2 As Excel.Worksheet
xlSheet2 = MyExcel.Sheets.Add()
xlSheet2.Cells(1, 1) = MyExcel.Worksheets("Sheet1").Cells(20, 12).Text
xlSheet2.Cells(1, 2) = MyExcel.Worksheets("Sheet1").Cells(20, 13).Text

MyExcel.Worksheets("Sheet1").Activate()
MyExcel.Worksheets("Sheet1").Range("F1:H23").Copy _
(MyExcel.Worksheets("Sheet2").Range("a200"))

MyExcel.Worksheets("Sheet2").Activate()

' New string List for paths
Dim pathList As New List(Of String)()
Dim eventList As New List(Of String)()

pathList.Add(MyExcel.Worksheets("Sheet2").Cells(1, 2).Text)

' Place "from to state" information on sheet2 @ A200
Dim z As Integer
For z = 1 To 22
    If MyExcel.Worksheets("Sheet2").Cells((z + 200), 1).Text = pathList(z) Then
        eventList.Add(MyExcel.Worksheets("Sheet2").Cells((z + 200), 3).Text)
    End If
Next

'Sort the "from to state" information
Dim sortRange As Range
sortRange = MyExcel.Worksheets("Sheet2").Range("A200:C223")
sortRange.Sort(Key1:=MyExcel.Worksheets("Sheet2").Range("A200:A223"), Order1:=XLSortOrder.xlAscending, _
    Key2:=MyExcel.Worksheets("Sheet2").Range("C200:C223"), Order2:=XLSortOrder.xlAscending, _
    Orientation:=XLSortOrientation.xlSortColumns, _
    Header:=XLYesNoGuess.xlYes)

Dim state As String
Dim lastState As String
Dim usedQueue As New List(Of String)
Dim unusedQueue As New List(Of String)

state = (xlSheet2.Cells(1, 2).Text & "1")
lastState = xlSheet2.Cells(2, 2).Text
xlWorkBook.Worksheets("Sheet2").Cells(8, 1) = "# of paths"
Recursive(MyExcel, xlWorkbook, state, lastState, 10, 201, 1, usedQueue, unusedQueue)
Appendix II

Function Recursive(MyExcel As Excel.Application, xlWorkbook As Workbook, state As String, lastState As String, _
    pathRowIndex As Integer, stateRowIndex As Integer, pathColumnIndex As Integer, usedQueue As List(Of String), _
    unusedQueue As List(Of String))

    Dim backStateTrue As Boolean
    Dim noUnusedBackToStateTrue As Boolean
    Dim eventList As New List(Of String)
    Dim backToState As String
    backToState = Nothing
    backStateTrue = False
    noUnusedBackToStateTrue = False
    Dim nextStateEvent As String
    nextStateEvent = Nothing
    Dim currentEvent As String
    currentEvent = Nothing
    Dim countRange As Range
    countRange = Nothing

    If Mid(state, 1, 3) = lastState Then
        'add last state to path
        'Reset lists and variables and increment column
        xlWorkbook.Worksheets("Sheet2").Cells(pathRowIndex, pathColumnIndex) = Mid(state, 1, 3)
        xlWorkbook.Worksheets("Sheet2").Cells(8, 2) = pathColumnIndex
        'unusedQueue.RemoveAt(unusedQueue.Count - 1)
        'look for unused paths in the queue/copy to queue
        'Dim backToState As String
        'No unused paths - Delete Column and Return

    If unusedQueue.Count = 0 Then
        Return DBNull.Value
    End If

    Dim rowIndexCount2 As Integer
backToState = unusedQueue(unusedQueue.Count - 1)
state = Mid(unusedQueue(unusedQueue.Count - 1), 1, 4)
rowIndexCount2 = CInt(Mid(backToState, 4, 1))

Dim g As Integer
    g = pathRowIndex
For g = 10 To pathRowIndex
    If (Mid(state, 1, 3) = xlWorkBook.Worksheets("Sheet2").Cells(g, pathColumnIndex).Text) Then
        rowIndexCount2 = rowIndexCount2 - 1
    End If
    If rowIndexCount2 = 0 Then
        pathRowIndex = g + 1
        Exit For
    End If
Next

backStateTrue = True

'Copy and paste to next path column
xlWorkBook.Worksheets("Sheet2").Range(xlWorkBook.Worksheets("Sheet2").Cells(pathRowIndex, pathColumnIndex), -
    xlWorkBook.Worksheets("Sheet2").Cells(10, pathColumnIndex)).Copy()

pathColumnIndex = pathColumnIndex + 1

Dim tempRange As Range
    tempRange = xlWorkBook.Worksheets("Sheet2").Cells(10, pathColumnIndex)
    tempRange.Select()
    xlWorkBook.Worksheets("Sheet2").Paste()

nextStateEvent = backToState

'clear event list
    eventList.Clear()
Else

'insert state into spreadsheet path and increment row index
xlWorkBook.Worksheets("Sheet2").Cells(pathRowIndex, pathColumnIndex) = Mid(state, 1, 3)
pathRowIndex = pathRowIndex + 1

'declare variables needed to get all events for current state
Dim w As Integer
Dim stateEvent As String
Dim firstCell As Range
Dim stateCount As Integer

stateCount = 0
firstCell = xlWorkBook.Worksheets("Sheet2").Cells(10, pathColumnIndex)
countRange = xlWorkBook.Worksheets("Sheet2").Range(firstCell, firstCell.End(X1Direction.xlDown))

'look for all events for current state
If Not backStateTrue Then
    For w = 201 To 223
        If xlWorkBook.Worksheets("Sheet2").Cells(w, 1).Text = Mid(state, 1, 3) Then
            stateCount = MyExcel.WorksheetFunction.CountIf(countRange, Mid(state, 1, 3))
            stateEvent = Mid(state, 1, 3) & stateCount & xlWorkBook.Worksheets("Sheet2").Cells(w, 3).Text
            eventList.Add(stateEvent)
        End If
    Next
End If

'check to see if state/event has been used
'if no unused paths exit
Dim c As Integer
Dim i As Integer

Dim midState As String
Dim unusedStateFound As Boolean

unusedStateFound = False
midState = Mid(state, 1, 3)

'this part removes loop elements from the eventList
'prior to checking for the next State/Event
'store states in eventHold
'Did this because of the remove shifts list
Dim eventHold As List(Of String)
Dim removeEvents As Boolean
removeEvents = False
eventHold = Nothing

For i = 0 To usedQueue.Count - 1
    If (Mid(usedQueue(i), 1, 3) = (midState)) Then
        Dim j As Integer
        For j = (eventList.Count - 1) To 0 Step -1
            If ((Mid(eventList(j), 1, 3) = (midState)) _
                And ((Mid(usedQueue(i), 5)) = (Mid(eventList(j), 5))) Then
                eventList.Remove(eventList(j))
            End If
        Next
    Next

'find next state
For c = 0 To eventList.Count - 1
    If Not (usedQueue.Contains(eventList(c))) Then
        nextStateEvent = eventList(c)
        unusedStateFound = True
        c = eventList.Count - 1
    End If
Next

'If there is no unused states for a current state
If Not unusedStateFound Then
    'No unused paths - Delete Column and Return
    If unusedQueue.Count = 0 Then
        xlWorkbook.Worksheets("Sheet2").Range(xlWorkbook.Worksheets("Sheet2").Cells(10, pathColumnIndex), _
                xlWorkbook.Worksheets("Sheet2").Cells(pathRowIndex, pathColumnIndex)).Delete()
    Return DBNull.Value
End If

Dim noUnusedBackToState As String
Dim rowIndexCount1 As Integer
Dim currentPathRowIndex1 As Integer

noUnusedBackToState = unusedQueue(unusedQueue.Count - 1)
state = Mid(unusedQueue(unusedQueue.Count - 1), 1, 4)
rowIndexCount1 = CInt(Mid(noUnusedBackToState, 4, 1))
currentPathRowIndex1 = pathRowIndex

Dim t As Integer
For t = 10 To pathRowIndex
    If (Mid(state, 1, 3) = xlWorkbook.Worksheets("Sheet2").Cells(t, pathColumnIndex).Text) Then
        rowIndexCount1 = rowIndexCount1 - 1
    End If
    If rowIndexCount1 = 0 Then
        pathRowIndex = t + 1
        Exit For
    End If
Next

noUnusedBackToStateTrue = True

xlWorkbook.Worksheets("Sheet2").Range(xlWorkbook.Worksheets("Sheet2").Cells(pathRowIndex, pathColumnIndex),
                                      xlWorkbook.Worksheets("Sheet2").Cells(currentPathRowIndex1, pathColumnIndex)).Clear()
nextStateEvent = noUnusedBackToState
End If
' add state event to used list
Dim queueCounter As Integer
queueCounter = usedQueue.Count - 1

If (backStateTrue Or noUnusedBackToStateTrue) = True Then
    While Not ((Mid(nextStateEvent, 1, 3)) = (Mid(usedQueue(queueCounter), 1, 3)))
        usedQueue.RemoveAt(queueCounter)
        queueCounter = queueCounter - 1
    End While
End If

usedQueue.Add(nextStateEvent)
state = Mid(nextStateEvent, 1, 4)
currentEvent = Mid(nextStateEvent, 5)

' select the next state
Dim v As Integer
Dim nextStateFound As Boolean
nextStateFound = False

For v = 201 To 223
    If ((xlWorkBook.Worksheets("Sheet2").Cells(v, 1).Text = Mid(state, 1, 3)) And _
        (xlWorkBook.Worksheets("Sheet2").Cells(v, 3).Text = currentEvent) And _
        Not nextStateFound) Then
        Dim countRangeNext1 As Range
        Dim firstCellNext1 As Range
        Dim stateCountNext1 As Integer
        Dim nextStateTemp As String

        stateCountNext1 = 0
        firstCellNext1 = xlWorkBook.Worksheets("Sheet2").Cells(10, pathColumnIndex)
        countRangeNext1 = xlWorkBook.Worksheets("Sheet2").Range(firstCellNext1, firstCellNext1.End(XLDirection.xlDown))
        countRangeNext1.Select()
        nextStateTemp = xlWorkBook.Worksheets("Sheet2").Cells(v, 2).Text
        stateCountNext1 = MyExcel.WorksheetFunction.CountIf(countRangeNext1, Mid(nextStateTemp, 1, 3))
        state = nextStateTemp & (stateCountNext1 + 1)
        nextStateFound = True
    End If
Next
End If

Next

Dim countRangeNext As Range
Dim firstCellNext As Range
Dim stateCountNext As Integer

If (backStateTrue Or noUnusedBackToStateTrue) = True Then
    unusedQueue.Remove(nextStateEvent)
ElseIf Not (Mid(state, 1, 3) = lastState) Then
    For r = 201 To 223
        If (xlWorkbook.Worksheets("Sheet2").Cells(r, 1).Text = Mid(nextStateEvent, 1, 3)) And _
            Not (xlWorkbook.Worksheets("Sheet2").Cells(r, 3).Text = currentEvent) And _
            Not (usedQueue.Contains(state & xlWorkbook.Worksheets("Sheet2").Cells(r, 3).Text)) Then
            Dim eventCount As Integer
            eventCount = CInt(Mid(nextStateEvent, 6))
            stateCountNext = 0
            firstCellNext = xlWorkbook.Worksheets("Sheet2").Cells(10, pathColumnIndex)
            countRangeNext = xlWorkbook.Worksheets("Sheet2").Range(firstCellNext, firstCellNext.End(xlDirection.xlDown))
            stateCountNext = MyExcel.WorksheetFunction.CountIf(countRangeNext, Mid(nextStateEvent, 1, 3))
            unusedQueue.Add(Mid(nextStateEvent, 1, 3) & (stateCountNext) & (xlWorkbook.Worksheets("Sheet2").Cells(r, 3).Text))
        End If
    Next
End If
'clear event list
eventlist.Clear()

'call the recursive function
Recursive(MyExcel, xlWorkbook, state, lastState, pathRowIndex, stateRowIndex, pathColumnIndex, _
    usedQueue, unusedQueue)

    Return DBNull.Value
End Function

End Module