Martin Tamayo
mt153@duke.edu

JFLAP Report

This document contains information on the following automata editors produced for OpenDSA:

- **Finite Accepter Editor**
- **Mealy Machine Editor**
- **Moore Machine Editor**

The code is maintained in the OpenDSA GitHub repository, which can be found at:

https://github.com/OpenDSA/OpenDSA

Instructions for using this repository can be found in the readme on the main page of the GitHub repository website. Note that special permissions are required in order to write to this repository, for which you will need to contact Dr. Clifford A. Shaffer at Virginia Tech University.

The most up-to-date versions of the editors can be found on the home page of my JFLAP Summer 2015 blog (scroll down to the bottom of the main page):

http://www.cs.duke.edu/csed/jflapodsa/martin/index.html

There exist two versions of the code – commented and code without comments. Both versions are maintained both on GitHub and on my blog. The commented code is maintained in separate folders and is not actually used for the automata editors that my blog links to. The commented code files simply exist to help users understand the code. Below is a list of every commented code file.

In **“automata commented”** folder:
- FAEditorCommented.js
- FATraversalCommented.js
- MealyEditorCommented.js
- MealyTraversalCommented.js
- MooreEditorCommented.js
- MooreTraversalCommented.js

In **“resources commented”** folder:
- CommandsCommented.js
- CustomPromptCommented.js
- serializableGraphCommented.js
- TraverseAccepterCommented.js
- TraverseTransducerCommented.js

Martin Tamayo
mt153@duke.edu

This document is 27 pages long. Below is a brief table of contents:

Martin Tamayo
mt153@duke.edu

References to other files within files:

It is important to note that in order for HTML pages / JavaScript code to work, it requires access to other files, which are called within the HTML markup / JavaScript code. Changing the locations of these files will **not** automatically update these pointers, meaning that you can accidentally break the code simply by changing the locations of files within a directory.

Therefore, if you want to change the file structure of the directory, you must go into each file and update the file paths of the other files they reference.

Here is a comprehensive list of every HTML document / JavaScript file I have created, as well as what other files they reference (and where to find those references):

FAEditor.html

- JSAV.css	Line 6
- CustomPrompt.css	Line 7
- FA.css	Line 8
- raphael.js	Line 48
- dagre.min.js	Line 49
- jquery.transit.js	Line 50
- JSAV.js	Line 51
- serializableGraph.js	Line 52
- underscore-min.js	Line 53
- FA.js	Line 54
- CustomPrompt.js	Line 55
- Commands.js	Line 56
- TraverseAccepter.js	Line 57
- FAEditor.js	Line 58

FAEditor.js

- FATraversal.html	Line 459
- grammarTest.html	Line 724
- conversionExercise.html	Line 730
- minimizationTest.html	Line 743

FATraversal.html

- JSAV.css	Line 6
- FA.css	Line 7
- raphael.js	Line 21
- dagre.min.js	Line 22
- jquery.transit.js	Line 23
- JSAV.js	Line 24
- serializableGraph.js	Line 25
- underscore-min.js	Line 26
- FA.js	Line 27
- TraverseAccepter.js	Line 28
- FATraversal.js	Line 29

FATraversal.js doesn't reference any other file.

MealyEditor.html

- JSAV.css	Line 6
- CustomPrompt.css	Line 7
- FA.css	Line 8
- raphael.js	Line 45
- dagre.min.js	Line 46
- jquery.transit.js	Line 47
- JSAV.js	Line 48
- serializableGraph.js	Line 49
- underscore-min.js	Line 50
- FA.js	Line 51
- CustomPrompt.js	Line 52
- Commands.js	Line 53
- TraverseTransducer.js	Line 54
- MealyEditor.js	Line 55

MealyEditor.js

- MealyTraversal.html	Line 516

MealyTraversal.html

- JSAV.css	Line 6
- FA.css	Line 7
- raphael.js	Line 21
- dagre.min.js	Line 22
- jquery.transit.js	Line 23
- JSAV.js	Line 24
- serializableGraph.js	Line 25
- underscore-min.js	Line 26
- FA.js	Line 27
- TraverseTransducer.js	Line 28
- MealyTraversal.js	Line 29

MealyTraversal.js doesn't reference any other file.

MooreEditor.html

- JSAV.css	Line 6
- CustomPrompt.css	Line 7
- FA.css	Line 8
- raphael.js	Line 45
- dagre.min.js	Line 46
- jquery.transit.js	Line 47
- JSAV.js	Line 48
- serializableGraph.js	Line 49
- underscore-min.js	Line 50
- FA.js	Line 51
- CustomPrompt.js	Line 52

- Commands.js	Line 53
- TraverseTransducer.js	Line 54
- MooreEditor.js	Line 55

MealyEditor.js

- MooreTraversal.html	Line 532

MealyTraversal.html

- JSAV.css	Line 6
- FA.css	Line 7
- raphael.js	Line 21
- dagre.min.js	Line 22
- jquery.transit.js	Line 23
- JSAV.js	Line 24
- serializableGraph.js	Line 25
- underscore-min.js	Line 26
- FA.js	Line 27
- TraverseTransducer.js	Line 28
- MooreTraversal.js	Line 29

MealyTraversal.js doesn't reference any other file.

The following files were not produced by me, but they do reference **FAEditor.html:**

- conversionExercise.html	Line 106
- minimizationTest.html	Line 400

The following files are resource files. They do not reference other files, but they are themselves referenced by a lot of the files I have created:

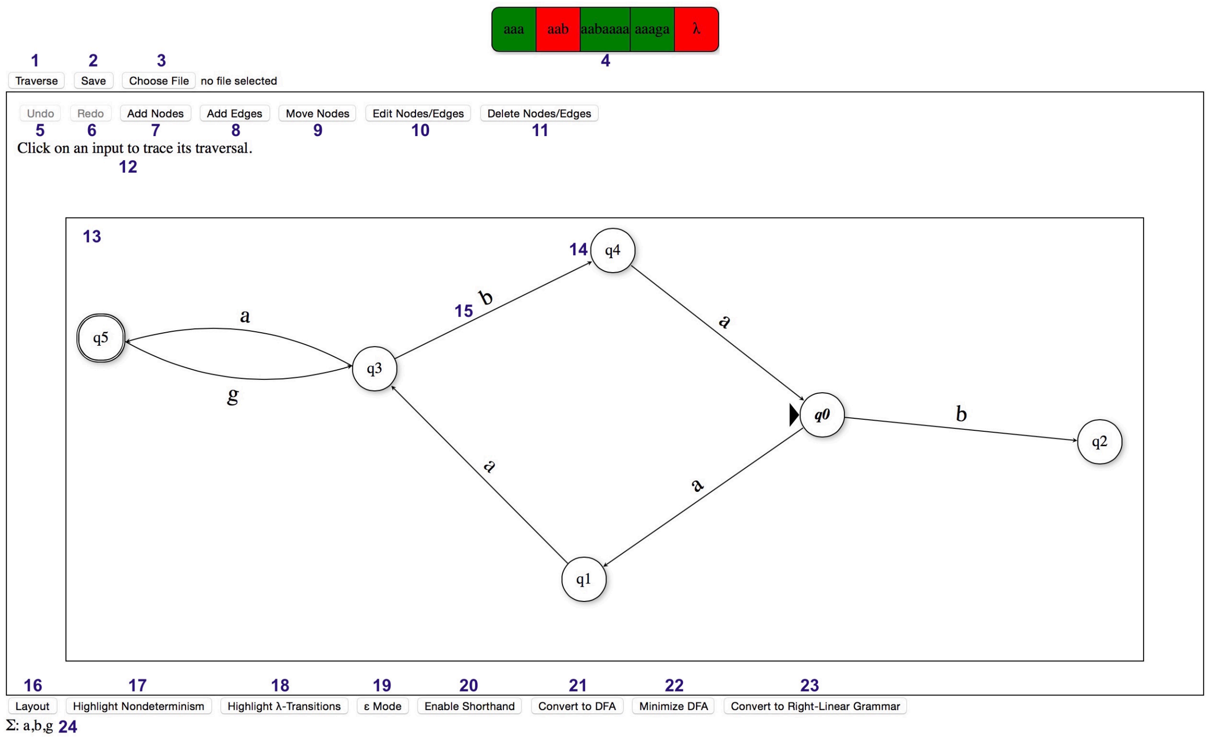
- Commands.js
- CustomPrompt.css
- CustomPrompt.js
- FA.css
- FA.js
- serializableGraph.js
- TraverseAccepter.js
- TraverseTransducer.js
- underscore-min.js

The following files come with JSAV, and are referenced by every HTML file I have made. JSAV is contained within a submodule in the OpenDSA GitHub repository, and my files on GitHub reference this location.

- JSAV.js
- JSAV.css
- dagre.min.js
- jquery.transit.js
- raphael.js

Martin Tamayo
mt153@duke.edu

Finite Accepter Editor



1. Traverse Button. Clicking here opens a prompt box for the user to enter input strings. The JSAV Array (4) will then be populated with these input strings.

2. Save Button. Clicking here reveals a link to download the FA in XML format. **Note that this button may have different effects depending on the browser being used. In Safari, it is necessary to right-click on the download link to save the file to the user's computer. The file has no default name or extension in Safari (however, since the XML markup is in JFLAP format, JFALP can still load the file just fine even without the .jff file extension).**

3. Load Button. Clicking here opens up the user's file directory. Loading an XML file that contains an FA will cause the FA to be initialized in the JSAV Graph (13). **Note that the button says "Choose File" because that text cannot be altered. This is a limitation of HTML5 – in terms of HTML objects, it is not a button, but rather an input field of type "file". Ideally, a way to somehow change this text to "Load File" should be coded. Also note that a new graph is loaded upon changing the file loaded with this button, meaning that if you try to reload a file from your computer that you already have loaded into the button, it will not work. (Not until you load a different file, then go back and load the original.) A workaround would be to clear the file loaded in the button after the graph from the file is initialized.**

4. JSAV Array. Contains the input strings entered by the user after clicking the Traverse button (1). Clicking on the array opens up a new window, in which the input string traversal on the graph is displayed.

5. Undo Button. Undoes the previous action. Disabled if the graph has not been edited.

6. Redo Button. Redoes the previous action if it was undone. Disabled if the Undo button (5) has not been clicked.

7. Add Nodes Button. Enables "Add Nodes" mode, in which clicking on the graph will cause a new node to appear on the graph.

8. Add Edges Button. Enables "Add Edges" mode, in which clicking two nodes on the graph in sequence will cause a new edge to appear between them.

9. Move Nodes Button. Enables "Move Nodes" mode, in which clicking a node followed by anywhere on the graph will move the node to that location on the graph.

10. Edit Nodes/Edges Button. Enables "Edit Nodes" mode, in which clicking a node will open a custom prompt box to configure the node, and clicking an edge label will open a custom prompt box to change the transition(s) along the edge.

11. Delete Nodes/Edges Button. Enables "Delete Nodes" mode, in which clicking a node will remove the node (and all connected edges) from the graph, and clicking an edge will remove the edge from the graph.

12. JSAV Message Output. Displays instructions to the user based on what the user clicks.

13. JSAV Graph. Displays the automaton. There are event handlers that fire upon clicking within the graph, depending on which editing mode is enabled (7-11).

14. JSAV Node. Automatically named / renamed as nodes are added / removed. There are event handlers that fire upon clicking on a node, depending on which editing mode is enabled (7-11). When a node is selected, it is highlighted yellow.

15. JSAV Edge / JSAV Edge Label. The edge label will display lambda/epsilon if it is the empty string. There are event handlers that fire upon clicking an edge or edge label, depending on which editing mode is enabled (7-11).

16. Layout Button. Repositions the nodes and edges on the graph by an automatic layout algorithm. Can be undone by clicking Undo (5).

17. Highlight Nondeterminism Button. If any nodes on the graph have outgoing lambda transitions or more than one of the same outgoing transition, they are highlighted blue.

18. Highlight Lambda Transitions Button. Any edges on the graph containing a lambda transition are enlarged and highlighted red.

Martin Tamayo
mt153@duke.edu

19. Epsilon Mode Button. Changes all instances of lambda on the graph to epsilon. When in epsilon mode, this button changes to the Lambda Mode Button, which has the opposite effect. This can also be undone by clicking Undo (5).

20. Enable Shorthand Button. Switches to shorthand mode, in which sequences of input symbols on an edge are permitted. When in shorthand mode, this button changes to the Disable Shorthand Button, which has the opposite effect. This can also be undone by clicking Undo (5). Also note that while shorthand mode is disabled, any edges with multiple input symbol transitions will be highlighted orange, and the Traverse button (1) will be disabled.

21. Convert to DFA Button. Opens a window (conversionExercise.html) to convert an NFA to an equivalent DFA. **This feature is incomplete – due to the lack of an algorithm to compare graph equivalence, the working DFA constructed in the new window cannot be compared to the model answer.**

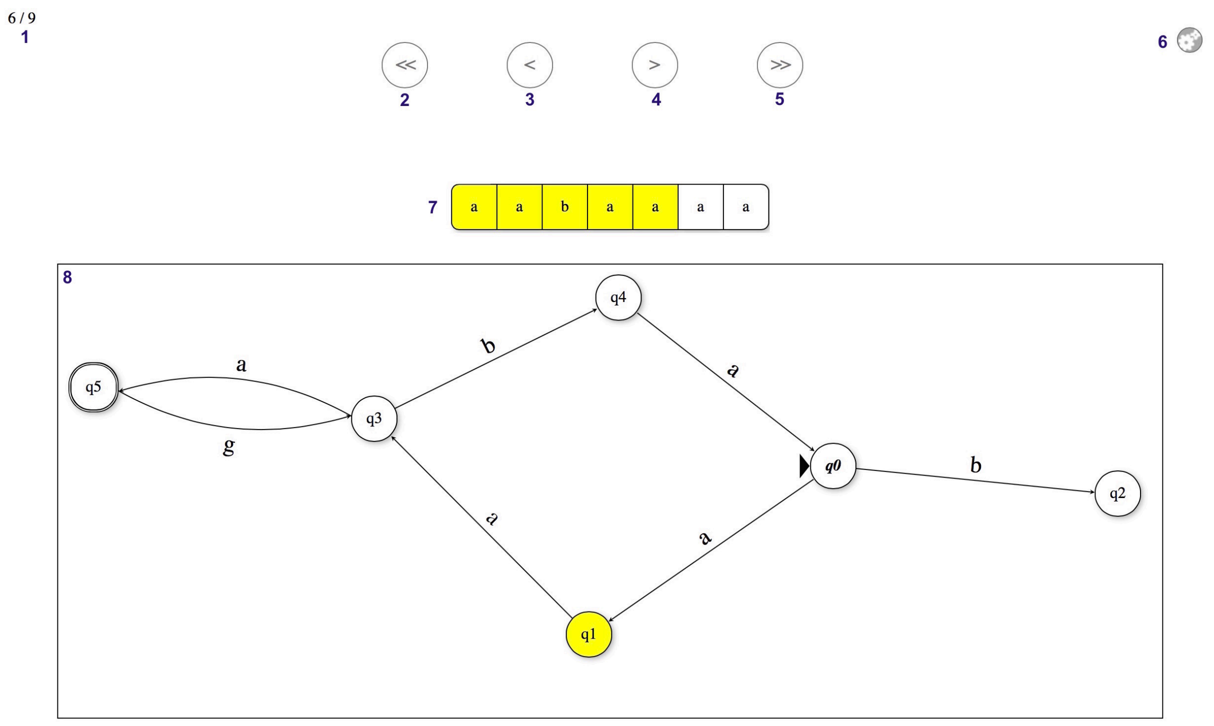
22. Minimize DFA Button. Opens a window (minimizationTest.html) to convert a DFA to an equivalent minimal-state DFA. Checks the graph for nondeterminism beforehand, **but does not check if the graph is a complete DFA. For this reason, the option to add a trap state to the graph to make a complete DFA should be implemented, as that would make this feature easier to use.**

23. Convert to Right-Linear Grammar Button. Opens a window (grammarTest.html) to convert an FA to an equivalent right-linear grammar. **Only works in certain browsers (confirmed working in Chrome, confirmed not working in Safari).**

24. Input Symbol Alphabet. Displays every input character accepted by a state in the automaton. Automatically updated as changes are made.

Martin Tamayo
mt153@duke.edu

Finite Accepter Traversal



1. JSAV Counter. Displays which slide the user is on in the slideshow.

2. JSAV Controls. This button takes the user to the beginning of the slideshow.

3. JSAV Controls. This button steps backward once in the slideshow.

4. JSAV Controls. This button steps forward once in the slideshow.

5. JSAV Controls. This button takes the user to the end of the slideshow.

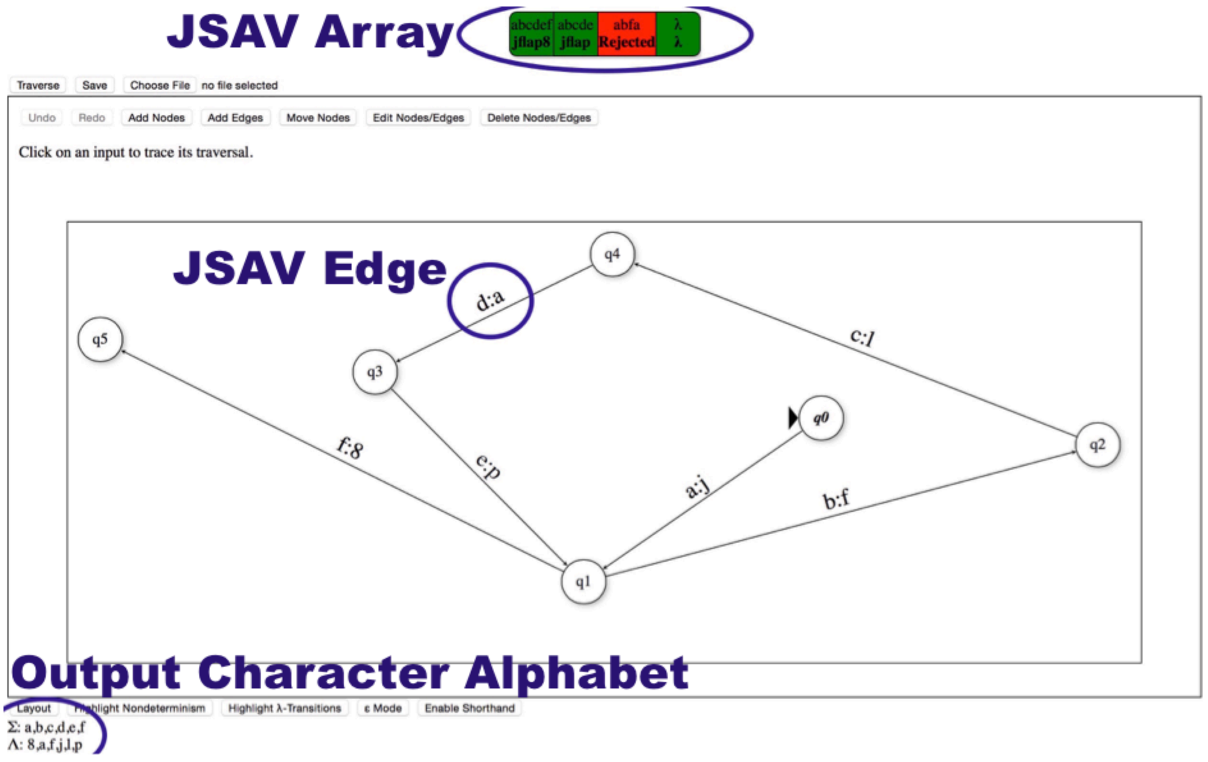
6. JSAV Settings. Opens a prompt where the user can change the animation speed.

7. JSAV Array. Displays the input string. The characters that have been traversed over are highlighted yellow. Clicking in the array will take the user to that step in the slideshow (where the corresponding input symbol has just been read). If the input string is accepted, the JSAV array is highlighted green in the last step of the slideshow. If the input string is rejected, the JSAV array is highlighted red in the last step of the slideshow.

8. JSAV Graph. Displays the graph as the algorithm traverses over it. The current node(s) are highlighted yellow, and if shorthand mode is enabled, any current edges are bolded (with the letter in the edge transition bolded where the traversal algorithm has reached). If the input string is accepted, all final states that the traversal finishes on in the last step of the traversal are highlighted green. If the input string is rejected, all current states in the last executable step of the algorithm are highlighted red.

Martin Tamayo
mt153@duke.edu

Mealy Machine Editor



The Mealy Machine Editor is mostly similar to the Finite Automaton Editor, though it is missing the "Convert to DFA", "Minimize DFA", and "Convert to Right-Linear Grammar" buttons. It also has a few extra features, a few of which are highlighted in this picture:

JSAV Array – Shows the output strings in bold beneath the input strings. If the input string is rejected, the output string displays "Rejected" (with red background). Again, clicking in this array opens a new window to view the traversal.

JSAV Edge – Edge transitions now support input characters and output characters, separated by a colon. These are entered separately in the edge prompt.

Output Character Alphabet – Displayed beneath the input character alphabet. Shows every unique output symbol the automaton can produce. Updated automatically as changes are made to the graph.

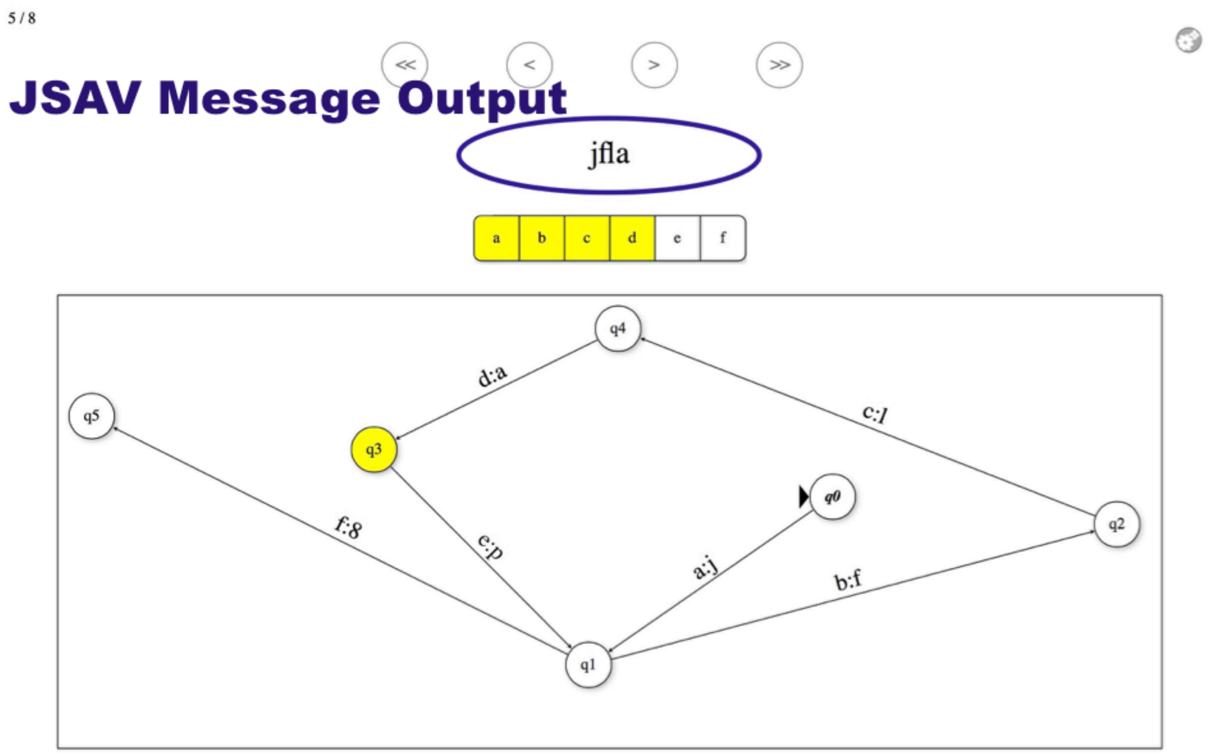
Clicking "Traverse" will run a check to see if the automaton is deterministic. If not, a warning is displayed, non-determinism and lambda transitions in the graph are highlighted, and the traversal prompt is not displayed.

Nodes cannot be final states, so this option is not offered when editing nodes.

The prompt box for adding / editing edges looks different, as each edge transition is represented by a pair of text fields – one for the input symbol and one for the output symbol. These are automatically concatenated with a colon when saving changes.

Martin Tamayo
mt153@duke.edu

Mealy Machine Traversal



The Mealy Machine Traversal window is identical to the Finite Automaton Traversal window for the most part. The major difference is that it runs a different traversal algorithm, since it is traversing a Mealy Machine and not a Finite Automaton.

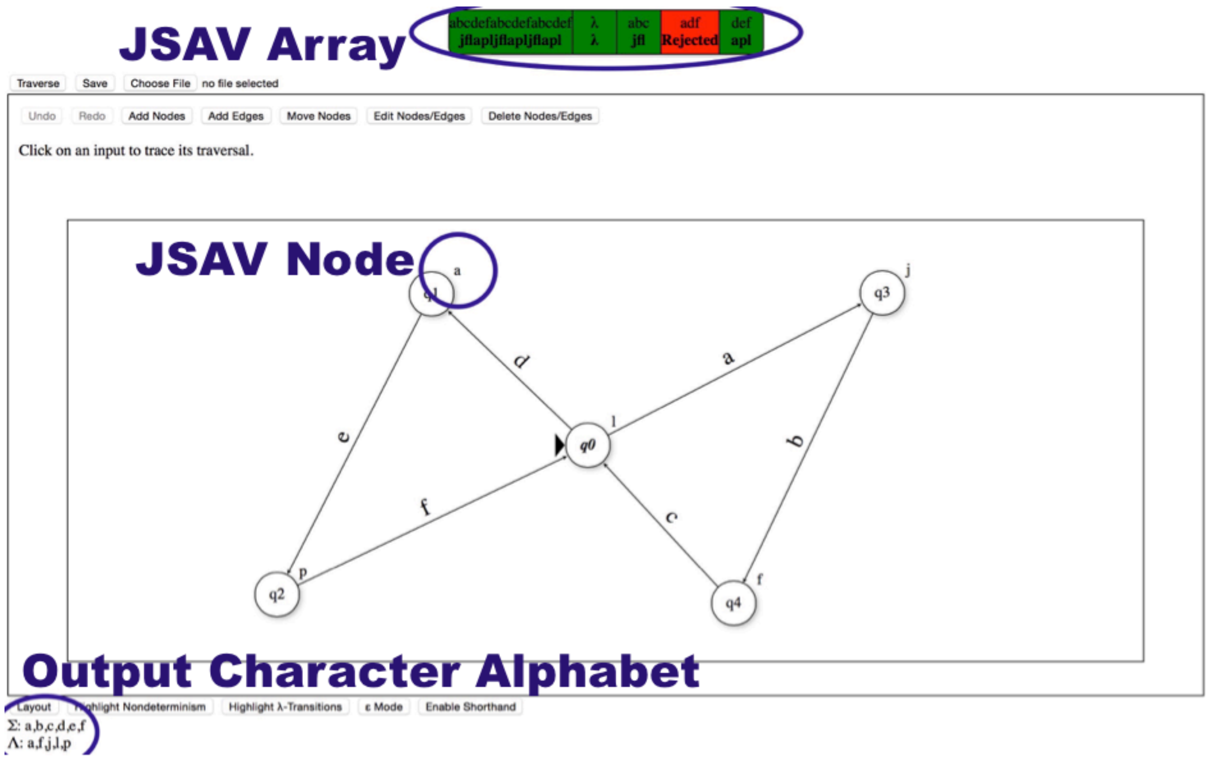
In terms of the interface, there is only one major difference:

JSAV Message Output – This is now centered above the JSAV Array, and displays the output string dynamically throughout the traversal. If the input string is rejected, the output string is replaced with "Rejected" in bold in the last step of the traversal. If the input string is accepted, but there is no output string, then "Accepted" is displayed in bold in the last step of the traversal. **Note that this screen shot is slightly outdated, in that the current version displays the output string below the input string as opposed to above it.**

Suggested Future Changes: *Add labels or other annotations next to the input array and output string that say "Input" and "Output" respectively, denoting that they are the input string and the output string. Another thing that could be done is displaying the output string as a JSAV array, so that it is aligned more intuitively with the input string.*

Martin Tamayo
mt153@duke.edu

Moore Machine Editor



The Moore Machine Editor is highly similar to the other two editors. Like the Mealy Machine Editor, it has some specific features:

JSAV Array – Shows the output strings in bold beneath the input strings. If the input string is rejected, the output string displays "Rejected" (with red background). Again, clicking in this array opens a new window to view the traversal.

JSAV Node – The output characters of the node are displayed next to them to the top right. These are added to the output string when the traversal reaches the node.

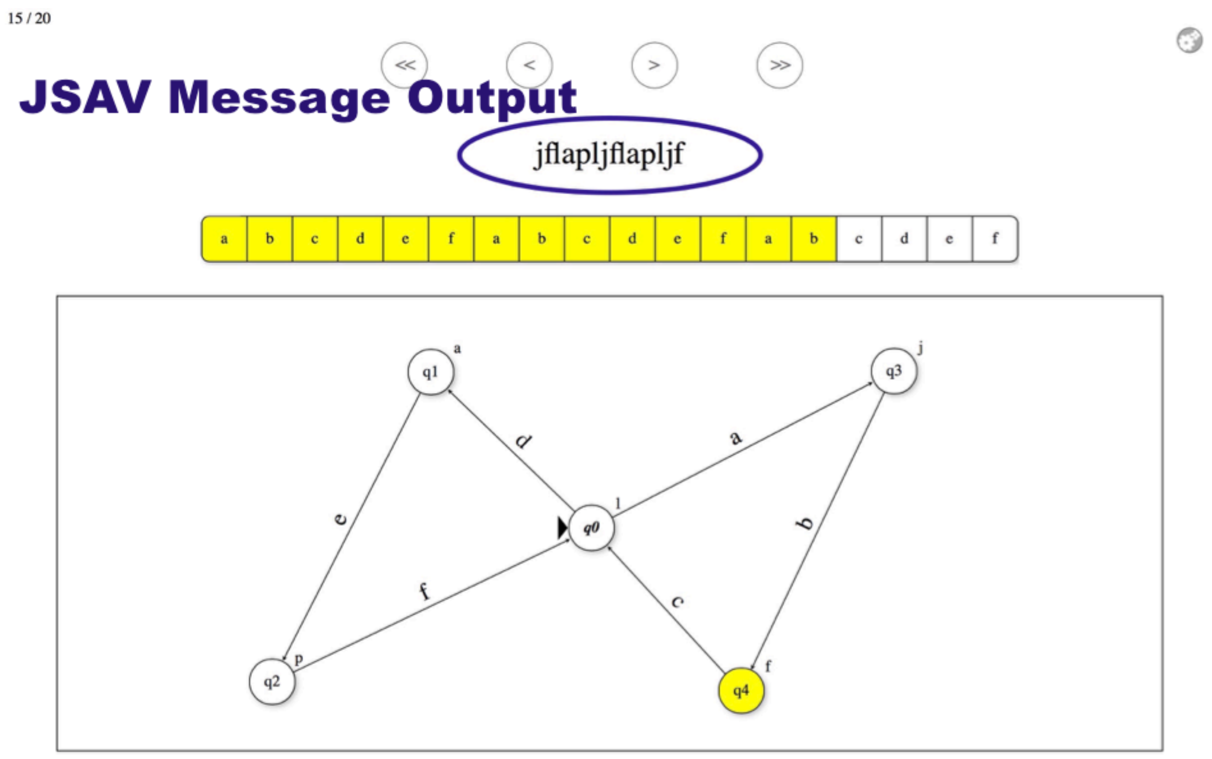
Output Character Alphabet – Displayed beneath the input character alphabet. Shows every unique output symbol the automaton can produce. Updated automatically as changes are made to the graph.

Clicking "Traverse" will run a check to see if the automaton is deterministic. If not, a warning is displayed, non-determinism and lambda transitions in the graph are highlighted, and the traversal prompt is not displayed.

The prompt box for editing nodes is now displayed whenever the user adds a new node, as the user is expected to provide an output character right away. This prompt box looks somewhat different for the Moore Machine Editor. Since nodes cannot be final states, this option is not offered. Additionally, there is a text field for entering the output character of the node. The output character can be the empty string – in this case, lambda (or epsilon) will be displayed next to the node.

Martin Tamayo
mt153@duke.edu

Moore Machine Traversal



The Moore Machine Traversal window is again mostly identical to the other two traversal windows. However, it does run its own traversal algorithms – ones that are specific to Moore Machines and not to Finite Automata or Mealy Machines.

Just like the Mealy Machine editor, the following difference should be noted:

JSAV Message Output – This is centered above the JSAV Array, and displays the output string dynamically throughout the traversal. If the input string is rejected, the output string is replaced with “Rejected” in bold in the last step of the traversal. If the input string is accepted, but there is no output string, then “Accepted” is displayed in bold in the last step of the traversal. **Note that, as with the Mealy Machine traversal window, this screen shot is slightly outdated, in that the current version displays the output string below the input string as opposed to above it.**

Suggested Future Changes: *As with the Mealy Machine traversal window, add labels or other annotations next to the input array and output string that say “Input” and “Output” respectively, denoting that they are the input string and the output string. Another thing that could be done is displaying the output string as a JSAV array, so that it is aligned more intuitively with the input string.*

Martin Tamayo
mt153@duke.edu

Future Directions and Suggested Changes

While the Finite Accepter Editor, Mealy Machine Editor, and Moore Machine Editor are fairly robust in their current form, there remain several known bugs and tools that still need to be incorporated. Below is a list of features that should be implemented next:

Add Trap State – There should be an option to add a trap state to a DFA in the Finite Accepter Editor. This instantly completes the DFA by adding a single node to the graph and connecting the other nodes to it with new edges. A complete DFA is necessary to run the minimization algorithm, making this feature desirable.

Moore to Mealy Conversion – There exists an algorithm to convert any Moore Machine into an equivalent Mealy Machine, though this has yet to be implemented in any version of JFLAP. It is worth creating, though note that there is no such algorithm to do the reverse (converting a Mealy Machine into a Moore Machine).

Various known bugs are noted throughout this document with **red, bolded text.** Below are some of the more prominent ones:

Convert to DFA / Minimize DFA – These are both features of the Finite Accepter Editor. Due to the lack of an algorithm to compare graph equivalence, the two files these buttons link to – conversionExercise.html and minimizationTest.html – do not work properly. In order for these to run properly (and open the converted DFA / minimized DFA in another FA Editor window), they need to successfully identify the student answer to be the same as the model answer.

Convert to Right-Linear Grammar – Another feature of the Finite Accepter Editor. The file the button links to – grammarTest.html – only works in certain browsers (not including Safari). Some efforts should be made to make grammars less browser-specific, so that more users can use them on their preferred browser.

Save Button – The Save button has slightly different effects depending on the browser. In Safari, it always saves the XML file as "Unknown", with no file extension. It would be ideal to have it download automatically as fa.jff (for example).

Load Button – This looks different in different browsers (particularly Firefox). Also note that the file added to the Load button should be cleared after the graph from the file has been loaded, or else the user will not be able to load that file again. It would also be ideal to somehow change the text from "Choose File" to "Load File" (or something similarly descriptive).

Input / Output Strings – The format of the Mealy / Moore Machine traversal windows could be improved, particularly by adding labels denoting that the JSAV array is the input string and the text displayed below it is the output string. The output string could also be reformatted so as not to awkwardly realign itself to the center of the page whenever it is changed.

Martin Tamayo
mt153@duke.edu

Editor JavaScript Files:

FAEditor.js / MealyEditor.js / MooreEditor.js

All three of these JavaScript files have very similar (and identically named) functions. Exceptions detailed at the end.

Note: *Instance variables* are italicized, **functions** are bolded.

jsav – Instance variable to store the JSAV algorithm visualization.

jsavArray – Instance variable to store the JSAV array.

first – Instance variable to store a pointer to the first node clicked in "Add Edges" mode. When a new edge is added, *first* and *selected* are used to identify which two nodes are connected by the edge.

selected – Instance variable to store a pointer to a node or edge that is clicked. Used to identify which node/edge should be edited/deleted in the various editing modes.

label – Instance variable to store a pointer to an edge label that is clicked. Used to identify which edge label to update when editing edges.

undoStack – Instance variable to store a stack of graphs added to whenever a change is made. The most recent graph from this stack is loaded whenever the "Undo" button is clicked. Note that at most 20 graphs may be stored on this stack at any time. The oldest graph will be removed if this limit is exceeded.

redoStack – Instance variable to store a stack of graphs added to whenever a change is undone. The most recent graph from this stack is loaded whenever the "Redo" button is clicked. Cleared whenever a change is made. Note that at most 20 graphs may be stored on this stack at any time.

g – Instance variable to store the JSAV graph.

lambda – Instance variable to store the JavaScript representation of lambda.

epsilon – Instance variable to store the JavaScript representation of epsilon.

emptystring – Instance variable to store which empty string representation is currently being used (lambda or epsilon).

willRejectFunction / pretraverseFunction – Instance variable to store a pointer to the appropriate traversal function to run (one allows multiple input symbols on an edge whereas one does not, depending on whether or not "Shorthand" mode is enabled). Called *willRejectFunction* in FAEditor.js and *pretraverseFunction* in MealyEditor.js and MooreEditor.js.

initialize – Initializes a graph with automatic layout. Calls the *initGraph* function.

initGraph – Initializes a graph by parsing a JSON representation. Mainly used by the *undo* / *redo* functions and when the graph is loaded by another window (conversionExercise.html, minimizationTest.html).

finalize – Updates the input character alphabet (and output character alphabet, if applicable) and calls the click handler functions for the JSAV graph.

checkEmptyString – Calls the *emptyString* function if the *initGraph* function loads a graph with a different empty string representation to the one currently being used. This allows the editor to load a graph that uses epsilon even if the editor is currently using lambda (and vice versa).

graphClickHandler – Sets click handlers for when the user clicks anywhere on the JSAV graph with "Add Nodes" or "Move Nodes" modes enabled.

nodeClickHandler – Sets click handlers for when the user clicks on a node with the "Add Edges" or "Move Nodes" or "Edit Nodes" or "Delete Nodes" modes enabled.

edgeClickHandler – Sets the click handler for when the user clicks on an edge with the "Delete Edges" mode enabled.

labelClickHandler – Sets the click handler for when the user clicks on an edge label with the "Edit Edges" mode enabled.

updateNode – Function called by custom prompt box when confirming changes on a node being edited. Calls the appropriate edit node function in Commands.js.

createEdge – Function to create an edge between two identified nodes with a specified edge label, by calling the *executeAddEdge* function in Commands.js.

updateEdge – Function to edit an edge with a specified edge label by calling the *executeEditEdge* function in Commands.js.

checkEdge – Checks a given edge for any transitions with more than one input symbol if "Shorthand" mode is disabled. Warns the user if the check returns positive.

checkAllEdges – Checks all edges for transitions with more than one input symbol if "Shorthand" mode is disabled. Warns the user if the check returns positive.

updateAlphabet – Updates the alphabet display at the bottom of the Editor view by checking all the input symbols on graph edges.

addNodes – Sets the editing mode to "Add Nodes", allowing the user to add new nodes to the graph. Triggered when the user clicks the "Add Nodes" button.

addEdges – Sets the editing mode to "Add Edges", allowing the user to add new edges to the graph between nodes. Triggered when the user clicks the "Add Edges" button.

moveNodes – Sets the editing mode to "Move Nodes", allowing the user to move nodes around on the graph. Triggered when the user clicks the "Move Nodes" button.

editNodes – Sets the editing mode to "Edit Nodes and Edges", allowing the user to change the properties of the nodes and edges on the graph. Triggered when the user clicks the "Edit Nodes/Edges" button.

deleteNodes – Sets the editing mode to "Delete Nodes and Edges", allowing the user to delete nodes and edges on the graph. Triggered when the user clicks the "Delete Nodes/Edges" button.

removeModeClasses – Clears all editing modes. Called by many other functions to disabled graph click handlers when they should no longer be active.

expandEdges – Enlarges edges to make them easier to click on. Used in "Delete Nodes and Edges" mode.

collapseEdges – Shrinks enlarged edges back to their standard size. Called whenever "Delete Nodes and Edges" mode is deactivated.

switchEmptyString – Switches the empty string representation being used (from lambda to epsilon and vice versa) by calling *emptyString*. Triggered when the user clicks the "Lambda/Epsilon Mode" button.

emptyString – Not to be confused with the *emptystring* instance variable. This function updates the graph whenever the empty string representation is changed, so if the user switches from lambda to epsilon, this function switches out all lambdas on the graph to epsilon, and vice versa.

updateTransitions – Helper function for the *emptyString* function that actually applies the changes in empty string representation to each graph edge (and, in the case of Moore Machines, each graph node as well).

testND – Examines every node on the graph and highlights them if they have any outgoing lambda transitions or more than one of the same transition leading to different nodes. Essentially indicates every instance of non-determinism on the graph, and returns true if the automaton is non-deterministic. This function can be triggered by the user by clicking the "Highlight Nondeterminism" button.

testLambda – Examines every edge on the graph and highlights them if they contain transitions on the empty string. This function can be triggered by the user by clicking on the "Highlight Lambda/Epsilon Transitions" button.

removeND – Removes CSS styles added to the graph by the *testND* and *testLambda* functions. Frequently called by other functions in order to keep the graph styling consistent.

layoutGraph – Repositions the nodes and edges on the graph automatically using a JSAV layout algorithm. Triggered when the user clicks on the "Layout" button.

readyTraversal – Disables all editing modes in preparation for graph traversal. Called upon return from the *TraversePrompt* custom prompt box.

displayTraversals – Renders the *TraversePrompt* custom prompt box for the user to specify input strings to simulate traversal on. Warns the user and quits if the graph lacks an initial state. (In the Mealy and Moore Machine editors, there are similar checks for non-determinism and whether or not the automaton is actually a Mealy or Moore Machine.) Triggered when the user clicks on the "Traverse" button.

traverseInputs – Runs upon returning from the TraversePrompt custom prompt box and invisibly traverses over the input strings, producing a JSAV array to display to the user which input strings were accepted or rejected (along with their respective output strings, in the case that the input strings were accepted by a Mealy or Moore Machine).

arrayClickHandler – Handler to load a traversal slideshow in a new window if the user clicks on an input string in the JSAV array. Called at the end of the *traverseInputs* function.

play – Serializes the graph and saves the graph and input string to local storage, then loads the graph in a new window and presents the traversal slideshow to the user. Called in response to the click handler on the JSAV array.

switchShorthand – Enables/disables shorthand mode by calling the *setShorthand* function. Triggered when the user clicks on the "Enable/Disable Shorthand" button.

setShorthand – Enables/disables shorthand mode, thus determining which traversal algorithms will be used. In shorthand mode, it is permissible to have a sequence of input symbols along one edge, and to progressively traverse over the edge one symbol at a time. If shorthand mode is disabled, yet there exist edges with sequences of input symbols on them, these edges are highlighted, the "Traverse" button is disabled, and a warning window is displayed to the user.

resetUndoButtons – Clears the *undoStack* and *redoStack* and disables the "Undo" and "Redo" buttons. Called whenever a new graph is loaded.

saveFAState / saveMealyState / saveMooreState – Saves the current state of the graph to the *undoStack* so that a simple call to the *undo* function can reload the graph in its original state. Called whenever the graph is changed by any operation. Also clears the *redoStack*.

undo – Pops a graph off the top of the *undoStack* and initializes it in the view, replacing the current graph (which is serialized and placed on top of the *redoStack*). Supports undoing actions. Triggered when the user clicks the "Undo" button. Note that this button will be disabled if the *undoStack* is empty.

redo – Pops a graph off the top of the *redoStack* and initializes it in the view, replacing the current graph (which is serialized and placed on top of the *undoStack*). Supports redoing undone actions. Triggered when the user clicks the "Redo" button. Note that this button will be disabled if the *redoStack* is empty.

onLoadHandler – Called upon page load to initialize the initial graph (by calling the *initialize* function itself). May load different graphs depending on what is saved in local storage (conversionExercise.html and minimizationTest.html may affect this).

serializeGraphToXML – Serializes the graph to XML format. Called by the *saveXML* function.

saveXML – Saves a graph and provides a link to download it in XML format. Triggered when the user clicks the "Save" button. **Note that this button may have different effects depending on the browser being used. In Safari, it is necessary to right-click on the download link to save the file to the user's computer. The file has no default name or extension.**

parseFile – Creates a graph from an XML representation in the form of a string and initializes it in the view (replacing any graph that was originally there). Called within the *waitForReading* function.

waitForReading – Function to trigger a call to the *parseFile* function once the FileReader is finished reading the loaded file. Called by the *loadXML* function.

loadXML – Function to load an XML file from the user's computer to display in the editor as a graph. Triggered when the user chooses a file from his/her local directory after clicking the "Choose File" button. **Note that this function runs whenever the form data is changed. What this means is that loading a file from the user's computer, then later attempting to load the same file will have no effect unless a different file was loaded between them.**

Martin Tamayo
mt153@duke.edu

Functions unique to FAEditor.js

convertToGrammar – Loads another window (grammarTest.html) to convert the FA to a right-linear grammar. Called when the user clicks the "Convert to Right-Linear Grammar" button. **Note that grammarTest.html does not work in Safari.**

convertToDFA – Loads another window (conversionExercise.html) to convert the FA to a DFA. Called when the user clicks the "Convert to DFA" button. **Does not check to see if the FA is already deterministic. Note that conversionExercise.html does not properly work because there is no function to compare graph equivalence, needed to compare the converted DFA to the model answer.**

minimizeDFA – Loads another window (minimizationTest.html) to convert the FA to a minimal-state DFA. Called when the user clicks the "Minimize" button. Does not proceed if the FA is currently non-deterministic. **Does not check to see if the FA is a complete DFA.**

Functions unique to MealyEditor.js

updateMealyOutput – Checks the Mealy Machine graph for every output character on an edge, and updates the output character display at the bottom of the view.

testMealy – Checks the Mealy Machine graph edges, and returns whether or not the graph is a valid Mealy Machine. Runs as a check before the traversal algorithm.

Functions unique to MooreEditor.js

doNothing – Placeholder function that does nothing. Passed to *MooreNodePrompt* to run when the user clicks "Cancel" if the user is editing an existing node.

cancelNode – Function that removes a newly-added node from the Moore Machine graph. Passed to *MooreNodePrompt* to run when the user clicks "Cancel" if the user is creating a new node.

createNode – Function to run upon creation of a new node in the Moore Machine editor. Passed to *MooreNodePrompt* to run when the user clicks "OK" if the user is creating a new node.

updateMooreOutput – Checks the Moore Machine graph for every output character on a node, and updates the output character display at the bottom of the view.

Martin Tamayo
mt153@duke.edu

Traversal JavaScript Files:

FATraversal.js / MealyTraversal.js / MooreTraversal.js

All three of these JavaScript files have very similar (and identically named) functions.

jsav – Instance variable to store the JSAV algorithm visualization.

arr – Instance variable to store the JSAV array.

g – Instance variable to store the JSAV graph.

lambda – Instance variable to store the JavaScript representation of lambda.

epsilon – Instance variable to store the JavaScript representation of epsilon.

initialize – Called when the page is loaded to initialize the graph and run the traversal input on it.

initGraph – Initializes the graph in the view from a serialized representation loaded from local storage. Called in the *initialize* function.

run – Creates the JSAV slideshow for the input string traversal on the graph. Called at the end of the *initialize* function.

runShorthand – Creates the JSAV slideshow for the input string traversal on the graph. Uses a traversal algorithm that allows edge transitions with sequences of input symbols on them. Called at the end of the *initialize* function if "Shorthand" mode was enabled in the editor.

arrayClickHandler – Handler to jump to a certain step in the traversal algorithm based on which index the user clicks on in the JSAV array (which displays the characters of the input string).

Martin Tamayo
mt153@duke.edu

Resource JavaScript Files:

Commands.js

executeAddNode – Adds a new node to a graph. Returns the node.

executeDeleteNode – Removes a node from a graph.

executeAddEdge – Adds a new edge to a graph between two nodes. Returns the edge.

executeDeleteEdge – Deletes an edge from a graph.

executeMoveNode – Moves a node to a new position on a graph.

executeEditFANode – Changes the properties of a node on a graph, including whether it is the initial state, whether it is a final state, and the label on the node. Only used by the Finite Automaton Editor.

executeEditMealyNode – Changes the properties of a node on a graph, including whether it is the initial state and the label on the node. Only used by the Mealy Machine Editor.

executeEditMooreNode – Changes the properties of a node on a graph, including whether it is the initial state, the label on the node, and the output character of the node. Only used by the Moore Machine Editor.

executeEditEdge – Changes the transition of an edge on a graph.

Martin Tamayo
mt153@duke.edu

CustomPrompt.js

renderBox – Sets up custom prompt box in view. Called by *render* functions in Prompt objects.

terminate – Removes custom prompt box from view. Called within functions in Prompt objects.

TraversePrompt – Custom prompt box for entering input strings to traverse on. Used by FA Editor, Mealy Editor, and Moore Editor.

- **render** – Sets up *TraversePrompt* in view. Calls *renderBox* function.
- **traverseInput** – Closes prompt box (by calling *terminate* function) and runs traversal function on the user inputs.
- **addNewInput** – Adds another text field to the prompt box for the user to add another input string.
- **deleteInput** – Removes a specified text field from the prompt box.

FANodePrompt – Custom prompt box for editing a node in FA Editor.

- **render** – Sets up *FANodePrompt* in view. Calls *renderBox* function.
- **ok** – Closes prompt box (by calling *terminate* function) and applies changes to the node being edited on the graph.

MealyNodePrompt – Custom prompt box for editing a node in Mealy Editor.

- **render** – Sets up *MealyNodePrompt* in view. Calls *renderBox* function.
- **ok** – Closes prompt box (by calling *terminate* function) and applies changes to the node being edited on the graph.

MooreNodePrompt – Custom prompt box for editing a node in Moore Editor.

- **render** – Sets up *MooreNodePrompt* in view. Calls *renderBox* function.
- **cancel** – Closes prompt box (by calling *terminate* function) and cancels changes to the node being edited on the graph.
- **ok** – Closes prompt box (by calling *terminate* function) and applies changes to the node being edited on the graph.

EdgePrompt – Custom prompt box for adding/editing an edge on a graph. Used by FA Editor and Moore Editor.

- **render** – Sets up *EdgePrompt* in view. Calls *renderBox* function.
- **addEdge** – Closes prompt box (by calling *terminate* function) and adds edge / applies changes to edge being edited on the graph.
- **addNewWeight** – Adds another text field to the prompt box for the user to add another edge transition.
- **deleteWeight** – Removes a specified text field from the prompt box.

MealyEdgePrompt – Custom prompt box for adding/editing an edge in Mealy Editor.

- **render** – Sets up *MealyEdgePrompt* in view. Calls *renderBox* function.
- **addEdge** – Closes prompt box (by calling *terminate* function) and add edge / applies changes to edge being edited on the graph.
- **addNewWeight** – Adds another pair of text fields to the prompt box for the user to add another edge transition.
- **deleteWeight** – Removes a specified pair of text fields from the prompt box.

serializableGraph.js

Node – Creates a JSON node object to add to the serialization.

Edge – Creates a JSON edge object to add to the serialization.

Graph – Creates a JSON graph object from the nodes and edges for the serialization.

serialize – Takes a graph and serializes it to JSON, calling *Node* to create JSON nodes from the graph nodes, *Edge* to create JSON edges from the graph edges, and finally *Graph* to complete the JSON representation of the graph. Returns the JSON graph.

lambdafy – Accepts an edge transition and changes every occurrence of lambda/epsilon to their respective HTML5 representations (which is necessary for serialization to JSON). Returns the updated edge transition.

lambdafyMoore – Accepts a Moore state output and, if it is the empty string, changes it from lambda/epsilon to their respective HTML5 representations (which is necessary for serialization to JSON). Returns the updated Moore state output.

delambdafy – Reverses the effects of *lambdafy*. Accepts an edge transition and changes every occurrence of lambda/epsilon to their respective JavaScript representations (which is necessary for rendering the graph in the view). Returns the updated edge transition.

delambdafyMoore – Reverses the effects of *lambdafyMoore*. Accepts a Moore state output and, if it is the empty string, changes it from lambda/epsilon to their respective JavaScript representations (which is necessary for rendering the graph in the view). Returns the updated Moore state output.

Martin Tamayo
mt153@duke.edu

TraverseAccepter.js

willReject – Takes a graph and an input string and returns whether or not the graph accepts the input string. Used by FA Editor.

willRejectShorthand – Takes a graph and an input string and returns whether or not the graph accepts the input string, using a traversal algorithm that allows edge transitions with sequences of input symbols on them. Used by FA Editor when "Shorthand" mode is enabled.

traverse – Calculates the current states in the next step of the traversal algorithm, given a graph, list of current states, and input symbol. Returns this list of next states.

pretraverseShorthand – Used by *willRejectShorthand* function to calculate the current states and edges in the next step of the traversal algorithm, given a graph, lists of current states and edges, and an input symbol. Returns these lists of current states and edges. Does not alter the CSS styling of the edges.

traverseShorthand – Used by FA Traversals view to calculate the current states and edges in the next step of the traversal algorithm, given a graph, lists of current states and edges, and an input symbol. Returns these lists of currents states and edges. Alters the CSS styling of the edges for purposes of displaying current edges in the view.

addLambdaClosure – Recursive function to determine the closure of a list of states. Called by all other NFA traversal algorithm functions. Returns an updated list to include all states reachable on lambda transitions from the original list of states.

Martin Tamayo
mt153@duke.edu

TraverseTransducer.js

pretraverse – Takes a Mealy Machine or Moore Machine graph and an input string and returns whether or not the graph accepts the input string, as well as the output string it produces.

pretraverseShorthand – Takes a Mealy or Moore Machine graph and an input string and returns whether or not the graph accepts the input string, as well as the output string it produces. Uses a traversal algorithm that allows edge transitions with sequences of input symbols on them. Only called when "Shorthand" mode is enabled in the Mealy Editor or Moore Editor.

traverseMealy – Given a Mealy Machine graph, current state, and input symbol, returns the next state in the traversal and the output symbol.

traverseMealyShorthand – Given a Mealy Machine graph, current state or edge, and input symbol, returns the next state or edge in the traversal and the output symbol. Only called when "Shorthand" mode is enabled in the Mealy Editor.

traverseMoore – Given a Moore Machine graph, current state, and input symbol, returns the next state in the traversal and the output symbol.

traverseMooreShorthand – Given a Moore Machine graph, current state or edge, and input symbol, returns the next state or edge in the traversal and the output symbol. Only called when "Shorthand" mode is enabled in the Moore Editor.