Technical Manual

State Table Generator (NextState)

Machine Problem 2

John Eron David Salongsongan
BS COMPUTER SCIENCE

Introduction

This guide is for the user to understand how the application works, and the methods within it.

How does a State Table be constructed?

To understand NextState, you should know how a State Table are built, as it is the basis of the algorithm.

For the construction of State Table, you will need the following:

1. Flip-flop type, and number of flip-flops on the circuit.
2. Number of inputs and outputs.
3. Equation of the Flip-flop inputs and output.

If you already have these, we can now start creating the table. The steps are as follows:

1. Create the table with the appropriate headings. The number of rows is equal to 2^n, where n is equal to the number of flip-flops plus number of inputs.

   For demonstration, we will construct the state table of a sequential circuit that has 1 JK flip-flop, 2 inputs B and C, and 1 output X. The number of rows in the table is $2^{1+2} = 8$.

   The table should look like as shown below.

Current State	Input		Next State	Output	Flip-Flop Inputs	
A	B	C	$A_{(t+1)}$	X	J_A	K_A

2. Fill up the current state column and input column with binary of 0 to 2^n. The state variables should hold the most significant bits.

   Using the example from above, we will fill up column with labels A, B, and C with the binary of 0 to 8.

   The table should look like as shown below.

Current State	Input		Next State	Output	Flip-Flop Inputs	
A	B	C	$A_{(t+1)}$	X	J_A	K_A
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

3. Using the values in the filled columns, fill up the flip-flop inputs and output by solving the expression given.

   For demonstration, we will be using the circuit earlier. The equation given to the flip-flop and output are as follows:

   - $J_A = A' + C$
   - $K_A = B'$
   - $X = ABC$

   We will plug in the values on the equations, so we can fill up the rows. After doing that, the table should look like as shown below.

Current State	Input		Next State	Output	Flip-Flop Inputs	
A	B	C	$A_{(t+1)}$	X	J_A	K_A
0	0	0		0	1	1
0	0	1		0	1	1
0	1	0		0	1	0
0	1	1		0	1	0
1	0	0		0	0	1
1	0	1		0	1	1
1	1	0		0	0	0
1	1	1		1	1	0

4. Using the flip-flop inputs, fill up the next state column. This will be based on the type of flip-flop used in the circuit. The behavior of each flip-flop is as follows:

T Flip-Flop		RS Flip-Flop			JK Flip-Flop		
T	Q(t+1)	S	R	Q(t+1)	J	K	Q(t+1)
0	Q(t)	0	0	Q(t)	0	0	Q(t)
1	Q(t)'	0	1	0	0	1	0
D Flip-Flop		1	0	1	1	0	1
D	Q(t+1)	1	1	?	1	1	Q(t)'
0	0						
1	1						

We will continue building the state table from before. Using the table of JK flip-flop, we will determine the next state of the circuit.

The table should look like as shown below.

Current State	Input		Next State	Output	Flip-Flop Inputs	
A	B	C	$A_{(t+1)}$	X	J_A	K_A
0	0	0	1	0	1	1
0	0	1	1	0	1	1
0	1	0	1	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	1	0	0	1	1
1	1	0	1	0	0	0
1	1	1	1	1	1	0

With this the state table is fully constructed and can now be used in building a state diagram.

NextState

After discussing how to build a state table, which is the basis of the algorithm of the program, we will discuss the program itself.

NextState is an application written in Java programming language. It implements the steps that are mentioned in the last section. The main task of the application is to generate a state table of a sequential circuit with a specific type and number of flip-flops, number of inputs, and number of outputs.

Features:

1. Users can choose the type of flip-flop to be used.
2. Users can use one to two flip-flops, one to two input variables, and up to one output variable.
3. Users can customize the state variables, input variables, and output variables.
4. The variables are not case sensitive, but this means that uppercase and lowercase of a variable will be considered as the same.
5. There is an in-app guide for the usage and operators that can be used.
6. There is a reset all button for easier deletion of all input.

Components:

The source folder of the application is composed of a package named “application” which contains the following:

1. Main.java
   Contains the main() method.
2. Stack.java
   An interface that implements the Stack data type.
3. LinkedStack.java
   Class that implements the Linked Stack.
4. Node.java
   Class that implements a Node, which is used in Stack.
5. InfixToPostfix.java
   Class that converts the infix notation of an expression into its postfix notation. It is used in the evaluation of the equations of the flip-flop inputs and outputs.

The “application” package also has sub-packages which are the following:

1. application.controller
   Contains the controller of the application. It consists of the following classes:
   1. MenuController.java

Class that serves as the controller of MenuView. It contains the methods that is executed when some action (i.e., clicking a button) is done on the menu. It also contains the methods that evaluate the expressions and builds the table.

2. application.exceptions
   Contains the exceptions that is thrown when an error arises during runtime. It consists of the following classes:
   1. DuplicateAssignmentException.java
      Thrown when there are same variables(letters) that are assigned to a different assignment.
   2. InvalidEquationException.java
      Thrown when the equation given by the user is deemed as invalid.
   3. InvalidVariableException.java
      Thrown when the variable assignment given by the user is invalid.
   4. MissingFieldException.java
      Thrown when not every field is filled up the user.
   5. StackEmptyException.java
      Thrown when stack is empty.
3. application.model
   Contains the data structure that is used in the application. It consists of the following classes:
   1. Data.java
      Contains the constructor for the data type Data, which is used to hold the information of each row of the state table.
4. application.view
   Contains the User Interface of the application. It consists of the following:
   1. MenuView.fxml
      Contains the structure of the UI.

Note: To better understand the list of methods, check the source code for reference.

Application Package

Package that contains all classes that builds the application

The package includes the following:

Main.java

Contains the main method of the application. Extends Application abstract class. This class contains the following methods:

1. public static void main() - launches the application.
2. public void start(Stage primaryStage) - loads the menu, through FXMLLoader, and shows it on screen. Overrides the start method on Application.

Node.java

Contains the constructor for the Node used in the Stack implementation. Adapted from the starter code of CMSC 123: Data Structures - Machine Problem 1: Stacks. This class contains:

1. private Object element – holds the element of the node.
2. private Node next – points to the next node of the current node.
3. public Node() – default constructor
4. public Node(Object element, Node next) – constructor
5. Object getElement() – getter of element
6. Node getNext() – getter of next
7. void setElement(Object element) – setter of element
8. void setNext(Node next) – setter of next

Stack.java

This is the Stack interface used in the program. Adapted from the starter code of CMSC 123: Data Structures - Machine Problem 1: Stacks. This interface contains:

1. public int size() – returns the number of elements in the stack
2. public boolean isEmpty() – returns true if the stack is empty, false otherwise
3. public Object top() – returns the element at the top of the stack. Throws StackEmptyException when stack is empty
4. public void push(Object element) – inserts element at the top of stack
5. public Object pop() – removes element at the top of the stack. Throws StackEmptyException when stack is empty

LinkedStack.java

A linked implementation of Stack data type. Adapted from the starter code of CMSC 123: Data Structures - Machine Problem 1: Stacks. This class implements Stack.java, thus all methods from the Stack interface is here. Other than those methods, this class contains:

1. private Node top – refers to the top of the stack
2. private int size – size of the stack
3. public LinkedStack() – default constructor

InfixToPostfix.java

A class used to convert infix expressions to postfix expression, which is used to evaluate the equations in constructing the table. This includes the following:

1. private String infix – variable that stores the infix expression
2. private String postfix - variable that stores the infix expression
3. public InfixToPostfix(String infix) – constructor. Sets infix as this.infix and this.postfix as an empty string.
4. public String convertToPostfix() – method that converts infix to postfix expression.
5. public static String format(String s) – method that formats a given string of expression
6. private static boolean isOperator(String x) – method that check if the String given is an operator. Returns true if it is one, otherwise returns false.
7. private int prec(Object a) – method that returns the defined precedence of the operators. The precedence is as follows: NOT > AND > XOR > OR.
8. private String popUntilOpenPar(Stack s) – method that performs pop operation on stack until the element to be popped is an open parentheses. Each popped element is added to a String that will be returned by the method after the operation.
9. private String finalAddOperator(Stack s) – method that empties the stack and adds the elements into a String, that will be returned.

There are also sub-packages in the Application package. These are the following:

Controller Sub-package

This sub-package contains the controller class of the Menu. This sub-package contains the following:

MenuController.java

This class is the controller of the Menu. This contains every method related to the user interface of the application. The class includes the following:

1. private ChoiceBox<String> ffType – drop-down list containing the flip-flop types that can be chosen
2. private ChoiceBox<Integer> numFF - drop-down list containing the number of flip-flops that can be used
3. private ChoiceBox<Integer> numIV - drop-down list containing the number of input variables that can be used
4. private ChoiceBox<Integer> numOV - drop-down list containing the number of output variables that can be used
5. private Button proceedButton – button “linked” to the “Proceed” button on the menu.
6. private TextField FlipFlop1 – text input component for the equation of a flip-flop input
7. private TextField FlipFlop2 – text input component for the equation of a flip-flop input
8. private TextField FlipFlop3 – text input component for the equation of a flip-flop input
9. private TextField FlipFlop4 – text input component for the equation of a flip-flop input
10. private TextField InputVar1 – text input component for the input variable assignment
11. private TextField InputVar2 – text input component for the input variable assignment
12. private TextField StateVar1 – text input component for the state variable assignment
13. private TextField StateVar2 – text input component for the state variable assignment

14. private TextField OutputVar – text input component for the output variable assignment
15. private TextField OutputEq – text input component for the equation of the output
16. private Button setButton – button “linked” to the “Set” button on the menu.
17. private Button resetButton – button “linked” to the “Reset All” button on the menu.
18. public void initialize() – initializes the menu, specifically the ChoiceBoxes, filling it with the choices. For ffType, the choices are T, D, RS, or JK. For numFF and numIV, the choice is either 1 or 2. Lastly, for numOV, the choice is either 0 or 1. The choices are added to the ChoiceBox.
19. private String FlipFlop – variable that stores the type of flip-flop chosen.
20. private int FlipFlopCount - variable that stores the number of flip-flops chosen.
21. private int InputCount - variable that stores the number of flip-flops chosen.
22. private int OutputCount - variable that stores the number of flip-flops chosen.
23. public void selectionMade() – this method is called when the button ‘Set’ is pressed. This method gets the value of each ChoiceBox and then enables and set to visible several TextFields, based on those values. If there are ChoiceBox that has empty value, MissingFieldException will be thrown and caught by a catch block, which then an error message will pop-up.
24. private List<String> varAssignments – variable that holds the variable assignments
25. public void proceedToTable() – this method is called when the button ‘Proceed’ is pressed. This method checks if every TextField has been filled up, if the variable assignments and equations are valid, and if there are no duplicate assignments. If something is detected, like duplicates, then an exception will be thrown and will be caught by the appropriate catch block, which then an error message will pop-up. If the fields passed the “tests”, then they will proceed through calling the methods fillTable(), and then createTable().
26. protected List<List<String>> Table – variable that holds each column (in form of List<String>) of the table.
27. protected List<String> Used – variable that holds the key (String) of used columns in the table.
28. private void fillTable() – this method is used to fill the “table”. First, every column, in form of List<String>, will be initialized as null, then depending on what choice the user made, these columns will be set as empty ArrayList. Then, the equations will be formatted properly, then converted into postfix, through the InfixToPostfix method. After that, based on the number of the input and states(flip-flops), the columns related to them will be filled. Then, these columns will be added to a List<List<String>> and the variable assignments will be added to another List<String>. After this, the equations will be solved by using the solve method. Next is filling up the next state columns by evaluating the flip-flop inputs. After this, each column will be added to Table if they are not null. A key associated to the column will also be added to Used at the same time.
29. private String capEq(String eq) – this method capitalizes every variable on the equation. This method also removes whitespace from the equation.
30. private List<String> solve(List<String> asgn, List<List<String>> vals, String eq, int size) – this method solves the equation given. This is done with stacks, specifically LinkedStack since the equation is now in postfix. First, if a variable is detected, the index of that variable is used to locate which list of values will be used, then the value at current index will be pushed to the stack. Else, the operation will be used on the latest two elements (except for NOT operation, which is only used on the latest element), which are popped, then the result will be pushed. Then, the result will be added to a List, which is then returned after every row is evaluated.

31. private boolean isStrAlpha(String a) – checks if the string given contains only alphabetic characters by splitting the string into an array of char then checking if every element is alphabetic. It returns true if that is the case, else it will return false.
32. public void createTable() – this method creates the TableView of the state table. First, the table will be initialized and the resize policy is set to constrained, which will maximize the window size. Next, table columns will be initialized, along with sub-columns which are then added if they are used (their keyword is found on the Used list). The columns are also set with specific minimum width, set as uneditable and unsortable. After this, based on the choices made in the ChoiceBox, their respective cell value factory is set with new property value factory based on their column position. After this, the data will be added to the table, then the columns. Then the StackPane will be initialized, and the table will be added to it, and finally be shown.
33. private ObservableList<String> getData() – this method constructs the Data class based on the size of Used list, and add each on these Data into the ObservableList<String> which are then returned after the operation.
34. private void resetAll() – method that is called when 'Reset All' button is pressed. A confirmation message will pop-up with two choices, and when 'OK' is pressed, resetInput() will be called to set all choices to null and input to empty strings.
35. private void resetInput() – method that set the choice in the drop-downs into null. resetUserTypedOnly() will be called after.
36. private void resetUserTypedOnly() – method the set the value of the text fields into empty string.
37. private String transform(String eq) – method that inserts AND operators on the equation, as the equation can contain this operator implicitly. This makes the operator present in the equation when it is transformed to postfix.
38. private boolean isVarValid(String a) – checks if the Variable (String a) is a valid variable assignment, valid meaning that it is composed of only one alphabetic character, when whitespace is removed. If the string does not fit the criteria, it returns false, else it will return true.
39. private boolean isEqValid(String e) – this method checks if the equation (in form of String) is valid based on the criteria defined. First, the whitespaces are replaced with an empty string so that each element in a string is "connected". It is also valid if the equation is "1" or "0", as this will signify that every value is 1 or 0. Then the equation will be split by character and will be checked one by one. If the character is an alphabetic character, it will check if it is a variable assignment, and will return false if it is not. If it is not alphabetic, it will check if it is operator and if it is not, it will return false. Then if the next character next to the operator is another operator, it will return false. Next is checking for presence of leading operators and trailing operators (except for NOT), and it will return false if there are one. If the equation passed all, then it will return true.
40. public String getFlipFlop() – getter of FlipFlop variable.
41. public void setFlipFlop(String flipFlop) – setter of FlipFlop variable.
42. public int getFlipFlopCount() – getter of FlipFlopCount variable.
43. public void setFlipFlopCount(int flipFlopCount) – setter of FlipFlopCount variable.
44. public int getInputCount() – getter of InputCount variable.
45. public void setInputCount(int InputCount) – setter of InputCount variable.

46. public int getOutputCount() – getter of OutputCount variable.
47. public void setOutputCount(int OutputCount) – setter of OutputCount variable.
48. public List<String> getVarAssignments() – getter of varAssignments variable.
49. public void setVarAssignments(List<String> varAssignments) - setter of varAssignments variable.
50. private Button guideButton – button "linked" to the "Guide" button on the menu.
51. private void guide() – method that is called when the Guide button is pressed. A information window will pop-up with two choices: Usage and Operator, which when either one will clicked, another information window will pop-up that includes information about the usage of the program and the operators available.

Exceptions Sub-Package

This sub-package contains the custom exceptions that is thrown during runtime. This sub-package contains:

DuplicateAssignmentException.java

This class extends Exception. This includes:

1. public DuplicateAssignmentException() – default constructor. Thrown when duplicate variable assignments is detected.

InvalidEquationException.java

This class extends Exception. This includes:

1. public InvalidEquationException() – default constructor. Thrown when the equation given by the user is deemed to be invalid.

InvalidVariableException.java

This class extends Exception. This includes:

1. public InvalidEquationException() – default constructor. Thrown when the variable assignment given by the user is deemed to be invalid.

MissingFieldException.java

This class extends Exception. This includes:

1. public InvalidEquationException() – default constructor. Thrown when the required input fields are not yet filled.

StackEmptyException.java

This class extends Exception. Adapted from CMSC 130: Data Structures – Machine Problem 1: Stacks. This includes:

1. public StackEmptyException(String err) – a constructor for the class. Thrown when the stack is empty when an operation is done to it (i.e., pop operation).

Model Sub-package

This sub-package contains the object that contains the data that is shown. Contains the following class:

Data.java

Class that holds the data on the rows of the table. Contains 11 different private SimpleStringProperty that will act as holder of data of a cell. These 11 have their own getter and setter. The class also contain a default constructor and constructors for 4, 5, 6, 7, 8, 9, 10, and 11 columns. The constructors accept String that will then be wrapped into SimpleStringProperty.

View Sub-package

This sub-package contains the view component of the application (GUI). Contains the following:

MenuView.fxml

A FXML file containing the information about the graphic design of the menu user interface.

If you want to edit the source code, you can use a text editor, or better yet an IDE, as it is needed to be compiled. Access the 'src' folder for the source code. This concludes this manual. Thank you for using NextState.