**Linux Command-line**

**Turing Machine**

**Simulator**

**Software Design**

Date: 26 April 2018

Author: Emmanuel Bonilla

Course: Computer Science 322, Software Engineering

**TABLE OF CONTENTS**

List of Figure Page 5 Revision History Page 6

1. Introduction Page 7
   1. Purpose Page 7
   2. Intended Audience Page 7
   3. Layout Page 7
2. Architecture Page 8
3. Data Dictionary Page 13
   1. Tape Class Page 13
      1. Description Page 13
      2. Association Page 13
      3. Attributes Page 13
      4. Methods Page 14
   2. InputAlphabet Class Page 15
      1. Description Page 15
      2. Association Page 15
      3. Attributes Page 15
      4. Methods Page 15
   3. TapeAlphabet Class Page 16
      1. Description Page 16
      2. Association Page 16
      3. Attributes Page 16
      4. Methods Page 16
   4. TransitionFunction Class Page 17
      1. Description Page 17
      2. Association Page 17
      3. Attributes Page 17
      4. Methods Page 17
   5. Transition Class Page 18
      1. Description Page 18
      2. Association Page 18
      3. Attributes Page 18
      4. Methods Page 19
   6. States Page 20
      1. Description Page 20
      2. Association Page 20
      3. Attributes Page 20
      4. Methods Page 20
   7. FinalStates Page 21
      1. Description Page 21
      2. Association Page 21
      3. Attributes Page 21
      4. Methods Page 21
   8. Main Page 22
      1. Description Page 22
      2. Association Page 22
      3. Attributes Page 22
      4. Methods Page 22
   9. Commands Page 23
      1. Description Page 23
      2. Association Page 23
      3. Attributes Page 23
      4. Methods Page 23
   10. Input String Page 24
       1. Description Page 24
       2. Association Page 24
       3. Attributes Page 24
       4. Methods Page 24
   11. Turing Machine Page 25
       1. Description Page 25
       2. Association Page 25
       3. Attributes Page 25
       4. Methods Page 26
4. User Interface Page 27
   1. Command Line Invocation Page 27
   2. Help Command Page 28
   3. Show Command Page 29
   4. View Command Page 31
   5. List Command Page 32
   6. Insert Command Page 33
   7. Delete Command Page 34
   8. Set Command Page 35
   9. Truncate Command Page 36
   10. Run Command Page 37
   11. Quit Command Page 38
   12. Exit Command Page 39
5. Files Page 40
   1. Turing Machine Definition File Page 40
   2. Input String File Page 41

References Page 42

**LIST OF FIGURES**

|  |  |
| --- | --- |
| Fig 2-1 Relationship Diagram | Page 7 |
| Fig 2-2 Turing Machine Diagram | Page 8 |
| Fig 2-3 Tape Class Diagram | Page 9 |
| Fig 2-4 Input\_Alphabet Class Diagram | Page 9 |
| Fig 2-5 Tape Alphabet Class Diagram | Page 9 |
| Fig 2-6 Transition\_Function Diagram | Page 10 |
| Fig 2-7 States Diagram | Page 10 |
| Fig 2-8 Final States Diagram | Page 10 |
| Fig 2-9 Transitions Diagram | Page 11 |
| Fig 2-10 Main Diagram | Page 11 |
| Fig 2-11 Commands Diagram | Page 11 |
| Fig 2-12 Input Strings Diagram | Page 12 |
| Fig 4-2 Valid Definition File | Page 40 |
| Fig 4-4 Input String Example | Page 41 |

**REVISION HISTORY**

|  |  |  |
| --- | --- | --- |
| *Description* | *Rational* | *Date* |
| Initial release specifications | Initial Release | 10 March 2018 |
| Update classes and descriptions | Second release | 22 April 2018 |
| Revised attributes and methods | Third release | 26 April 2018 |

**1.0 INTRODUCTION**

* 1. **Purpose**

This document is designed to work in conjunction with the requirements specification sheet. For the Linux command-line Turing machine simulator use to development the source code.

* 1. **Intended Audience**

The main audience for this document is developers or maintainers of the application in an academic environment.

* 1. **Layout**

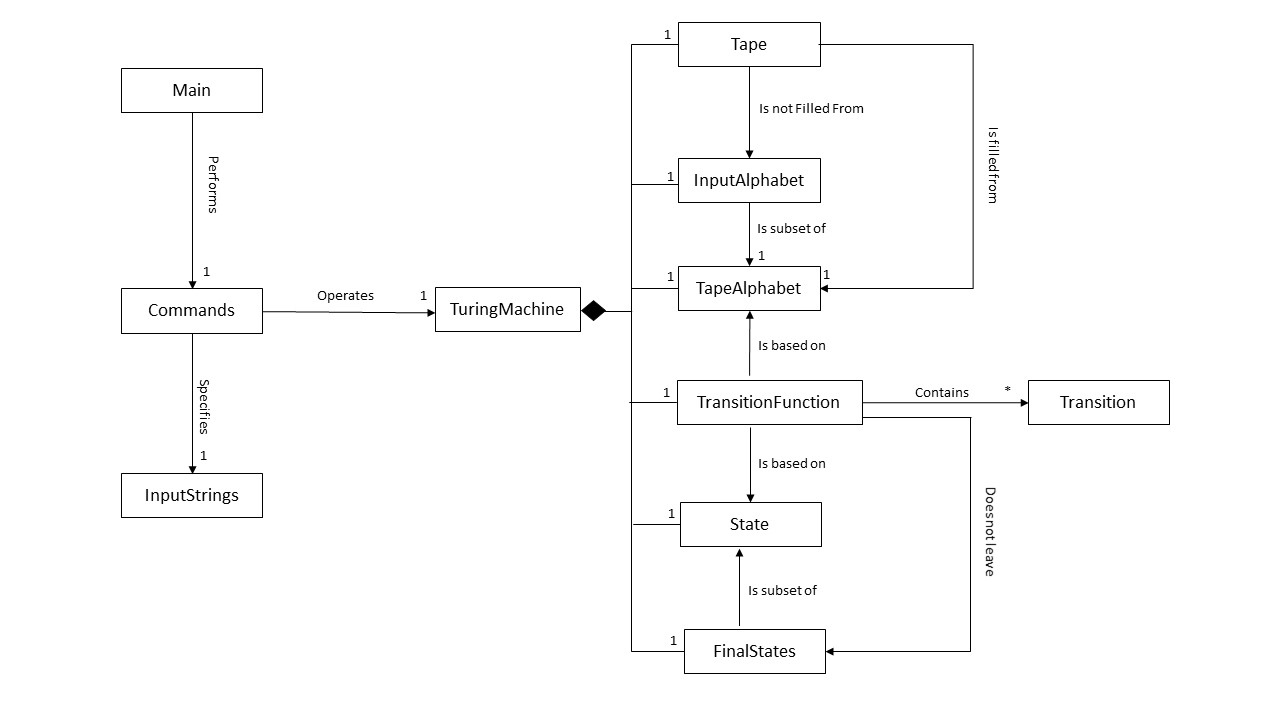
***Architecture:*** Displays object-oriented software with diagrams of each class.

***Data Dictionary:*** Describes the reason for the Turing Machine application. Explain the hardware and software environment for the development and installation of the application. Also, how to run the application, and interactively trace the operation of a

Turing Machine.

***User Interface:*** Describes the input and output devices for the Turing Machine. As well the Turing Machine definition file, input string file, with descriptions of each.

***Files***: Describes how to use the Turing Machine application from beginning to end of operation. Including detailed descriptions of commands and settings.

**2.0 ARCHITECTURE**

|  |
| --- |
| Figure 2-1 Relationship Diagram |

|  |
| --- |
| Fig 2-2 TuringMachine Diagram |
| Attributes |
| description: String\_Vector  initialState: String  currentState: String  originalInputString: String  numberOfTransition: Integer  valid: Boolean  used: Boolean  operating: Boolean  accepted: Boolean  rejected: Boolean |
| Methods |
| turingMachine (In definitionFileName: String): Void  viewDefinition (): Void  viewInstantaneousDescription (In maximumNumberOfCells: Integer): Void  initialize (In inputString: String): Void  performTransitions (In maximumNumberOfTransitions: Integer): Void  terminateOperation (): Void  inputString (): String  totalNumberOfTransitioins (): Integer  isValidDefinition (): Boolean  isValidInputString (In value: String): Boolean  isUsed (): Boolean  isOperating (): Boolean  isAcceptedInputString (): Boolean  isRejectedInputString (): Boolean |

|  |
| --- |
| Fig 2-3 Tape Class Diagram |
| Attributes |
| cells: String = “ “  currentCell: Integer = 0  blankCharacter: Character = ‘ ‘ |
| Methods |
| load (In Out Definition: file, In Out valid: Boolean): Void  validate (In Out inputAlphabet: InputAlphabet, In Out tapeAlphabet: TapeAlphabet, In Out valid: Boolean): Void  view (): Void  initialize (In inputString: String): Void  update (In writeCharacter: Char, In moveDirection: Direction): Void  left (In maximumNumberOfCells: Integer): String  right (In maximumNumberOfCells: Integer): String  currentCharacter (): Character  isFirstCell (): Bool  Tape (): Tape |

|  |
| --- |
| Fig 2-4 InputAlphabet Class Diagram |
| Attributes |
| Alphabet: Character \_Vector = {} |
| Methods |
| element (In index: Integer): void  load (In Out Definition: definition, In Out valid: Boolean): Void  size (): Integer  view (): Void  isElement (In value: Character): Boolean |

|  |
| --- |
| Fig 2-5 TapeAlphabet Class Diagram |
| Attributes |
| Alphabet: Character \_Vector = {} |
| Methods |
| isElement (In value: Character): Boolean  load (In Out Definition: definition, In Out Valid: Boolean): Void  view (): Void |

|  |
| --- |
| Fig 2-6 TransitionFunction Diagram |
| Attributes |
| transitions: String\_Vector = {} |
| Methods |
| load (In Out Definition: File, In Out valid: Boolean): Void  destinationState (In index: Integer): String  Find\_Transition (In sourceState: String, In readCharacter: Character, In out destinationState: String, Out writeCharacter: Character, Out moveDirection: Direction): Boolean  isSourceState (In state: string): Boolean  readCharacter (In index: integer): Character  size (): Integer  sourceState (In index: integer): Character  view (): Void  writeCharacter (In index: integer): Character |

|  |
| --- |
| Fig 2-7 States Diagram |
| Attributes |
| names: String\_Vector = {} |
| Methods |
| load (In Out Definition: file, In Out valid: Boolean): void  view (): void  isElement (In value: String): Boolean |

|  |
| --- |
| Fig 2-8 FinalStates Diagram |
| Attributes |
| names: String\_Vector = {} |
| Methods |
| load (In Out file: Definition, In Out valid: Boolean): Void  view (): Void  isElement (In value: String): Boolean  element (In index: Integer): String  size (): Integer |

|  |
| --- |
| Fig 2-9 Transitions Diagram |
| Attributes |
| read: Character  write: Character  source: String  destination: String  move: Direction |
| Methods |
| Transition (In sourceState: String, In readCharacter: Character, In definitionState: String, In writeCharacter: Character, In moveDirection: Direction): Transition  sourceState (): State  readCharacter (): Character  destinationState (): String  writeCharacter (): Character  moveDirection (): Direction |

|  |
| --- |
| Fig 2-10 Main Diagram |
| Attributes |
|  |
| Methods |
| main (): Integer |

|  |
| --- |
| Fig 2-11 Commands Diagram |
| Attributes |
| inputStrings: String\_Vector |
| Methods |
| startTuringMachine (In filename: String): Void |

|  |
| --- |
| Fig 2-12 InputStrings Diagram |
| Attributes |
| inputStrings: String\_Vector  overwrite: Boolean  loaded: Boolean |
| Methods |
| InputString (): InputString  size (): Integer  element (In index: Integer): String  isElement (In value: String): Boolean  load (In value: String): Void  view (): Void  insert (In value: String): Void  remove (In inputStringToDelete: String): Void  writeListToFile (In filePath: String): Void |

**3.0 Data Dictionary**

**3.1 Tape Class**

**3.1.1 Description**

The tape class for the Turing machine consist of an ordered sequence starting at Current\_cell (Fig 2-3) set to zero. At the beginning of the operation the input string is stored in the lowest number tape cell and so forth. The character contained in the Current\_cell has a combination of options read / write or move left / right.

**3.1.2 Associations**

There are two associations for the Tape class “is filled from” and “is not filled from”.

Is filled form is used to validate the blank character for initialization of the Turing machine tape on the *tapeAlphabet* (Fig 2-5).

Is not filled form is used to is used to validate the clank character for initialization of the Turing machine tape on the *inputAlphabet* (Fig 2-4).

**3.1.3 Attributes**

There are three attributes in the tape class that can be found in fig (2-3).

* 1. cells: String “ ”, is a dynamically allocated string that contains the Turing machine tape and may be extended by appending the blank character.
  2. currentCell: Integer = 0, is the index of the current cell on the Turing Machine Tape.
  3. blankCharacter: Character = ‘ ’, is used to initialize and extend the Turing Machine Tape.

**3.1.4 Methods**

*load (In Out definition: file, In Out valid: Boolean): void* method reads from the definition file. An error message will display if the character found is a reserved character or the next keyword does not follow.

*validate (In Out inputAlphabet: InputAlphabet, In Out tapeAlphabet: TapeAlphabet, In Out valid: Boolean): void* method checks if the blank character is the tapeAalphabet. It will display an appropriate error message if it is in the inputAlphabet.

*initialize (in inputString: string): void* method sets the cells to the inputString plus the blankCharacter, and sets currentCell to zero.

*view (): void* method will display the character of the Turning Machine

*update (In writeCharacter: character, in moveDirection: direction): void* method will write a character to the current cell location, then move the cell left or right. Update method will then increment or decrement the index.

*left (In maximumNumberOfCells: Integers): string* method will return a string of characters left of the current cell from the Turing Machine tape not including the current cell. If the string is truncated, the reserved character ‘<’ will be inserted to the front of the string.

*right (In maximumNumberOfCells: Integers): string* method will return a string of characters right of the current cell from the Turing Machine tape not including the current cell. If the string is truncated, the reserved character ‘>’ will be truncated.

*currentChar (): Character* method returns the character in the current cell on the Turing Machine Tape.

*isFirstCell (): Boolean* method will only return true if the current cell is in the zero index.

*Tape (): Tape* Is the constructor that sets the attributes to default, cells (“ “), currentCell(0), blankCharacter (‘ ‘).

**3.2 inputAlphabet Class**

**3.2.1 Description**

The inputAlphabet class is existing as a part of the Turing Machine. It is designed to read from the definition file to validate is it from the inputAlphabet and collect all the characters that are part of the language.

**3.2.2 Associations**

There are two associations for the inputAlphabet class “is subset of” and “is not filled from”.

*Is subset of* is used to validate the input alphabet contained inside the tapeAlphabet.

*Is not filled form* is used to validate the blank character for initialization of the Turing machine tape on the inputAlphabet (Fig 2-4).

**3.2.3 Attributes**

*Alphabet: Character\_Vector = {}* is a character vector that holds all the input characters of the language from the definition file.

**3.2.4 Methods**

*Load (In Out Definition: File, In Out Valid: Boolean)* method is designed to read in from the definition file. As well to return a true or false if the character is reserved, not printable, or the next keyword does not follow. It will display and appropriate message upon an error.

*Validate (In Out Valid: Boolean)* method is designed to check if the input alphabet is in the tape alphabet. Return true if valid, false if not valid with an appropriate error message.

*View ()* method will display the character of the Turning Machine

*isElement (In Value: Character): Boolean* method is designed to return true if the character is compared and found in the input alphabet. If not found return false

**3.3 tapeAlphabet Class**

**3.3.1 Description**

The tapeAlphabet class is designed to read from the definition file, validate the characters, view them, and determine if the character is in the input alphabet.

**3.3.2 Associations**

There are two associations for the inputAlphabet class “is subset of” and “is not filled from”.

*Is subset of* is used to validate the input alphabet contained inside the tapeAlphabet.

*Is not filled form* is used to validate the blank character for initialization of the Turing machine tape on the inputAlphabet (Fig 2-4).

**3.3.3 Attributes**

*Alphabet: Character\_Vector = {}* is a character vector that holds all the input characters of the language from the definition file.

**3.3.4 Methods**

*load (In Out Definition: file, In Out Valid: Boolean)* method is designed to read in from the definition file. As well to return a true or false if the character is reserved, not printable, or the next keyword does not follow. It will display and appropriate message upon an error.

*view ()* method will display the character of the Turning Machine

*isElement (In value: Character): Boolean* method is designed to return true if the character is compared and found in the input alphabet. If not found return false

**3.4 transitionfunction Class**

**3.4.1 Description**

The Transition Function Class is designed to look at all the transitions in the definition file. It will interpret the transition based on state, and current character. Also validate the transition with the tape alphabet.

**3.4.2 Associations**

The Transition Function class has a “is based on” association with the tapeAlphabet and States. With the tapeAlphabet it is used to validate the character of the current cell. With the States it is used to validate the state loaded in from the definition file. It is also used to create a set of transitions for the Turing machine application.

**3.4.3 Attributes**

Attributes here up to the developer’s implementation not denoted by associations.

**3.4.4 Methods**

*load (In Out Definition: file, In Out valid: Boolean)* method is designed to read in from the definition file. As well to return a true or false if the character is reserved, not printable, or the next keyword does not follow. It will display and appropriate message upon an error.

*destinationState (In index: Integer): String* method is designed to return the attribute destination.

*Find\_Transition (In sourceState: String, In readCharacter: Character, In out destinationState: String, Out writeCharacter: Character, Out moveDirection: Direction): Boolean* method is designed to find the transition based on the source state and in read character. Also write the character, move the direction and return true or false based on what transition is found. If and error, it will display and appropriate message.

*validate (In state: string): Boolean* method is designed to check if the input alphabet is in the tape alphabet. Return true if valid, false if not valid with an appropriate error message.

*view (): Void* method will display the character of the Turning Machine

*readCharacter (In index: integer):* *Character* method returns the attribute read.

*writeCharacter (In index: integer): Character* method returns the attribute write.

*size (): Integer* method returns the attribute size

*sourceState (In index: integer): String* method checks to see if the index is out of bounds.

**3.5 Transition Class**

**3.5.1 Description**

The Transition Class will instruct the Turing Machine Application which direction to move the tape, which character to write and what state to move to.

**3.5.2 Association**

The class Transition is contained in the class Transition\_Function, receiving messages to it by the Transition function.

**3.5.3 Attributes**

*Source: String* defines the state that the Turing machine must be in to use this transition.

*Read: Character* defines what the Turing machine must read to use this transition.

*Destination: String* denotes the state the Turing machine should advance in the use of this transition.

*Write: Character* denotes the character that the Turing machine should write to the current cell if the Turing machine uses this transition.

*Move: Direction* denotes which direction the Turning machine should move the tape in the process of using this transition.

**3.5.4 Methods**

*sourceState (): String* method sourceState returns the attribute source.

*readCharacter (): Character* method readCharacter returns the attribute read.

*desitinationState (): String* method destinationState returns the attribute destination.

*writeCharacter (): Character* method writeCharacter returns the attribute write.

*moveDirection (): Direction* method moveSirection returns the attribute move.

*Transition (In sourceState: String, In readCharacter: Character, In writeCharacer: Character, In moveDirection: Direction)* method transition will be considered as the constructor of the transition class, because it will initialize source, read, write, move.

**3.6 State Class**

**3.6.1 Description**

States class will determine if another string is a state or not a state, and show the current states to the user of the application.

**3.6.2 Associations**

*Does not leave* with finalStates class express that the Turing Machine cannot leave while in the final state.

*Is based on* with Transition\_Function to validate the transitions in the function.

**3.6.3 Attributes**

*Names: String\_Vector = {}* is vector of string type that will store all individual states from the definition file

**3.6.4 Methods**

*load (In Out Definition: file, In Out valid: Boolean): Void* method is designed to read in from the definition file. As well to return a true or false if the character is reserved, not printable, or the next keyword does not follow. It will display and appropriate message upon an error.

*view ()* method will display the character of the Turning Machine

*isElement (In value: String): Boolean* method takes a string value and returns true or false base on all the elements in the list.

**3.7 finalStates Class**

**3.7.1 Description**

FinalStates class is designed to hold a subset of states and recognize only work with the Turing Machine class.

**3.7.2 Associations**

*Does not leave* with the FinalStates is designed to stop the Turing Machine from continuing of the input string is in an accepting state.

**3.7.3 Attributes**

*Names:String\_Vector = {}* is designed to hold strings that denote the final states.

**3.7.4 Methods**

*load (In Out file: Definition, In Out valid: Boolean)* method is designed to read in from the definition file. As well to return a true or false if the character is reserved, not printable, or the next keyword does not follow. It will display and appropriate message upon an error.

*view ()* method will display the character of the Turning Machine.

*isElement (In value: String): Boolean* method takes a string value and returns true or false base on all the elements in the list.

*element (In index: Integer): Boolean* method returns the attribute names.

*size (): Integer* method returns the attribute size

**3.8 Main Class**

**3.8.1 Description**

The Main class is designed to initialize the Turing Machine at startup.

**3.8.2 Associations**

Runs with the class commands to start the user interface on the Turing Machine.

**3.8.3 Attributes**

No attributes

**3.8.4 Methods**

*main (): Integer* method will run when invoked from the command line.

**3.9 Command Class**

**3.9.1 Description**

The Command class is designed for user interface to input commands for the Turing Machine.

**3.9.2 Associations**

*Performs* with Main class to receive messages from main.

*Specifies* with the Input\_String to initialize the .str file from the current working directory.

*Operates* with Turing Machine depending on the user input and valid files.

*Assigns* with Configuration\_Settings to initialize the default settings.

**3.9.3 Attributes**

*Commands: string* will define all possible commands from the user.

**3.9.4 Methods**

*Start (In definition: String)* method is designed to initialize Input\_strings, Commands, Turing\_Machine, and start the user interface that will allow the user to run command

**3.10 inputString Class**

**3.10.1 Description**

The inputString Class is designed to load the .str file from the current working directory.

**3.10.2 Associations**

Specifies from the command method the .str file.

**3.10.3 Attributes**

*Inputs: String\_Vector = {}* is vector of string type that receives messages by command method.

**3.10.4 Methods**

*startTuringMachine (In fileName: String): Void* method is start the Turing Machine and give the user options to run the machine.

**3.11 Turing Machine Class**

**3.11.1 Description**

The turningMachine class is the main program that will determine if the Turing machine definition file is valid, input string is valid, is accepted or rejected by the language and be able to determine its current state.

**3.11.2 Associations**

The association *uses* denotes that the turingMachine class will be receiving messages delegated to it by the Command class.

**3.11.3 Attributes**

*Description: String\_vector* stores the definition file.

*initialState: String* stores the initalState.

*currentState: String* stores the current state of the Turing machine definition file.

*originalInputString: String* stores the original initialized input string.

*numberTransitions: Integer* stores the total number of transitions the Turing machine has performed on the input string.

*valid: Boolean* determines if the .def file is valid.

*used: Boolean* determines if the Turing machine has been used.

*operating: Boolean* determines if the Turing Machine is working on a input string.

*finalState: Boolean* checks if the Turing Machine is in the final state.

**3.11.4 Methods**

*TuringMachine (In definitionFileName: String): TuringMachine* method to initialize its private members, attempt to open .def file.

*viewDefinition (): Void* method is designed to display the entire description of the Turing Machine based on the .def file.

*viewInstantaneousDescription (In maximimNumberOfCells: Integer): Void* method displays the current description of the tape.   
*initialize (In InputString: String): Void* method stores the input string into the Original\_inputString.

*performTransition (In maximumNumberOfTransition: Integer): Void* method will take in a number of transition to be performed by the user.

*terminateOperation (): Void* method will reset the Turing Machine to default and set the state to not operating.

*inputString (): String* method will return originalInputString attribute.

*totalNumberOfTransitions (): Integer* Integer method will return the attribute numberOfTransitions.

*isValidDefinition (): Boolean* method determines if the .def file is Valid.

*isValidInputString(In Value: String): Boolean* method determine if the input string contains only characters from the Input\_Alphabet.  
*isUsed ():Boolean* method will return the attribute used.

*isOperating (): Boolean* method determine if the Turing machine is working on an input string.

*isAcceptedInputString (): Boolean* checks if the current state is accepted.

*isRejectedInputString (): Boolean* checks if the current state rejected.

1. **USER INTERFACE**

**4.1 Command Line Invocation**

To run the Turing Machine simulator, the user must enter the definition file name and input file name. Files need to be in the working directory. For a valid input, the user must enter file and input name without any extension (.def or .str). Also, a blank space must be entered between the definition file name and input file name. Please refer to Fig 3-1 for an example

**$./TM Test**

TM is the name of the application. Test is the name of the definition file. If the files names are valid the simulator will read both files and load them to the application. Once complete, a display will prompt the user for success or failure. Only if successful will the application continue, and display “command: “.

**4.2 Help Command**

Invocation: h or H (only)

*Help* user command will display each of the eleven commands with a brief description of each purpose. No other command from the user is required. The application will return to command prompt after displaying Fig 3-2.

**Command: H**

Once invoked the command line will display.

**(D)ELETE: Delete input string from list**

**E(X)IT: Exit application**

**(H)ELP: Display Commands**

**(I)NSERT: Insert input string**

**(L)IST: List input strings**

**(Q)UIT: Quit operation of TM on input string**

**(R)UN: Run TM on input string**

**S(E)T: Set maximum number of transitions to perform**

**SHO(W): Show status of application**

**(T)RUNCATE: Set truncation length of instantaneous description**

**(V)IEW: View Turing Machine**

**4.3 Show Command**

*Show* user command is designed to display nine different categories related to the application. There are two main details status of the application and configuration settings. Refer to Fig 4-3 for an example.

**Command: W**

Once invoked the command line will display.

**Course: Computer Science 322**

**Semester and Year: Fall, 2018**

**Instructor: Dr. Neil Corrigan**

**Author: Emmanuel Bonilla**

**Version: 1.0**

**Configuration Setting: Max Transitions: 1**

**Max Left and Right cells: 1**

**TM Name: TM**

**Status of TM: TM has never run on input string**

If the Turing Machine has completed the operation on a input string it will display one of three options on the Status of TM: line.

**Option 1:**

**Last Input String: AABABABBBA**

**Result: Accepted**

**Total Transitions: 87**

**Option 2:**

**Last Input String: AABA**

**Result: Rejected**

**Total Transitions: 3**

**Option 3:**

**Last Input String: AABABABBBA**

**Result: Terminated**

**Total Transitions: 87242**

**4.4 View Command**

Invocation: v or V (only)

*View* user command is designed to display the definition of the currently loaded Turing Machine on the monitor in a form readable by the user. Fig 5-6 is an example of a form readable to the user.

**Command: V**

Once invoked the command line will display.

**Q = {s0, s1, s2, s3, s4}**

**Σ = {a, b}**

**Γ = {a, b, X, Y, -}**

**δ(s0, a) = (s1, X, R)**

**δ(s0, Y) = (s3, Y, R)**

**δ(s1, a) = (s1, a, R)**

**δ(s1, b) = (s2, Y, L)**

**δ(s1, Y) = (s1, Y, R)**

**δ(s2, a) = (s2, a, R)**

**δ(s2, X) = (s0, X, R)**

**δ(s2, Y) = (s2, Y, R)**

**δ(s3, Y) = (s3, Y, R)**

**δ(s3, -) = (s4, -, R)**

**q0 = s0**

**B= -**

**F = {s4}**

**4.5 List Command**

Invocation: l or L (only)

*List* user command is designed to display a list of input strings upon which Turing Machine may be operated. Each input string will appear on its own line. The numbers will start at 1 and continue sequentially. If the list is empty, the application will produce an appropriate error message.

**Command: L**

Once invoked the command line will display.

**[1] ababa**

**[2] bbb**

**[3] aa**

**[4] \**

If there is an empty list the command line will display.

**No input strings**

**4.6 Insert Command**

Invocation: i or I (only)

*Insert* input string command allows users to enter an input string and append it to the list. The application will prompt the user for an input string. For the string to be valid it must be check three conditions. First if all characters are in the input alphabet (Σ). Second, ensure the string is not already in the list. Third, there are no additional white spaces. The application will display an appropriate message weather the input is valid or invalid.Once invoked the command line will display one of 3 options.

Option 1 (Valid input string):

**Command: I**

**Please enter an Input String: Test**

**Success: Test has been inserted**

Option 2 (Invalid input string):

**Command: I**

**Please enter an Input String: Test**

**Error: Input string is invalid**

Option 3 (Input string already exists):

**Command: I**

**Please enter an Input String: Test**

**Error: Input string Already exists**

**4.7 Delete Command**

Invocation: d or D (only)

*Delete* input string command allows the user to delete an input string from list. Input string is indicated by the number only, not by typing the string. If user entry is valid, string will be deleted, and list will be renumbered, even if the string is being evaluated. The application will display an appropriate message whether the input is valid or invalid. Once invoked the command line will display one of 3 options.

Option 1 (Valid Index):

**Command: D**

**Enter index to be deleted: 5**

**Success: BABABB has been deleted**

Option 2 (Invalid Index):

**Command: D**

**Enter index to be deleted: 7**

**Error: No input string with an index 7**

Option 3 (Not integer or negative integer):

**Command: D**

**Enter index to be deleted: -3**

**Error: Invalid input string index**

**4.8 Set Command**

Invocation: e or E (only)

*Set* transitions command allows the user to change the setting for maximum number of transitions on an input string. Valid necessary input can be a positive integer or blank space. Blank space will be treated as empty and transition value will not change. Invalid inputs are negative numbers, decimals, or other characters. The application will display an appropriate message whether the input is valid or invalid. Once invoked the command line will display one of 3 options.

Option 1 (Positive Integer):

**Command: E**

**Configure Setting: Maximum Number of Transitions**

**Current Setting: [1]**

**Enter New Value: 9**

**Success: Maximum number has been changed to 9**

Option 2 (Negative Integer):

**Command: E**

**Configure Setting: Maximum Number of Transitions**

**Current Setting: [1]**

**Enter New Value: 9**

**Error: Only positive integers allowed**

Option 3 (Blank Space):

**Command: E**

**Configure Setting: Maximum Number of Transitions**

**Current Setting: [1]**

**Enter New Value: 9**

**Success: Current setting has not been changed**

**4.9 Truncate Command**

Invocation: t or T (only)

*Truncate* command allows the user to change the setting for the maximum number of cells to the left and right of the tape head to display an ID. A valid users input can be a positive integer or blank space. Blank space will be treated as empty and transition value will not change. Invalid inputs are negative numbers, decimals, or other characters. The application will display an appropriate message whether the input is valid or invalid. Once invoked the command line will display one of 3 options.

Option 1 (Positive Integer):

**Command: T**

**Configure Setting: Maximum # of cells, left and right of tape head**

**Current Setting: [11]**

**Enter New Value: 5**

**Success: Maximum # of cells changed to 5**

Option 2 (Negative Integer):

**Command: T**

**Configure Setting: Maximum # of cells, left and right of tape head**

**Current Setting: [11]**

**Enter New Value: -7**

**Error: Only positive integers allowed**

Option 3 (Positive Integer):

**Command: T**

**Configure Setting: Maximum # of cells, left and right of tape head**

**Current Setting: [11]**

**Enter New Value:**

**Success: Current setting has not been changed**

**4.10 Run Command**

Invocation: r or R (only)

*Run* command allows the user to trace operation of Turing Machine on an input string using ID’s. The application prompts the user for a positive string number. The string number represents the string to be operated in the application. If user input is not in the list or not a positive integer an appropriate message will appear. If valid input application will display message. Once invoked the command line will display one of 3 options.

Option 1 (Valid input string):

**Command: R**

**Enter Input String Index: 57**

**Instantaneous Description: [56] [s0] ABABBB**

**Total Amount of Transitions: [56]**

**Accepted: Input string ABABBB has been accepted**

Option 2 (Invalid input string):

**Command: R**

**Enter Input String Index: 57**

**Instantaneous Description: [56] [s0] ABABBB**

**Total Amount of Transitions: [56]**

**Rejected: Input string ABABBB has been rejected**

Option 3 (Blank Space):

**Command: R**

**Enter Input String Index:**

**Instantaneous Description: [0] [s0] ABABBB**

**4.11 Quit Command**

Invocation: q or Q (only)

*Quit* command allows the user to terminate operation of Turing Machine on input string before completion. If the Turing Machine is operating on a string, a message will be displayed. The message will display “input string”, if the Turing Machine has accepted or rejected the input string, and the number of transitions performed before quitting. An appropriate error message will be displayed if Turing machine is not operating on a string. Once invoked the command line will display one of two options.

Option 1 (Initialized Turing Machine):

**Command: Q**

**Rejected: input string BABB is not accepted or rejected**

**Total amount of transitions: 2**

Option 2 (Uninitialized Turing Machine):

**Command: Q**

**Error: Turing Machine Unitialized**

**4.12 Exit Command**

Invocation: x or X (only)

*Exit* command allows the user to terminate execution of application. If the input string was changed in anyway, the entire list is written to the input string file, replacing the original file. A display message will appear if the file was successful or a failure. Once invoked the command line will display one of two options.

Option 1 (Success):

**Command: X**

**Success: Test file has been created**

Option 2 (Failure):

**Command: X**

**Error: Test file failed to be created**

1. **Files**
   1. **Turing Machine Definition File**

The Turing Machine Definition File is a free file format that describes the finite state automata that a user would like test input strings on. Using this definition file will allow the reusability of the application and allow it to adapt any language. An example of this Turing machine definition file:

|  |
| --- |
| Fig 4-2 Valid Definition File |
| This Turing machine accepts the language of one or more a’s followed by the same number of b’s  STATES: s0 s1 s2 s3 s4  INPUT\_ALPHABET: a b  TAPE\_ALPHABET: a b X Y –  TRANSITION\_FUNCTION:  S0 a s1 X R  S0 Y s3 Y R  S1 a s1 a R  S1 b s2 Y R  S1 Y s1 Y R  S2 a s2 a R  S2 X s0 X R  S2 Y s2 Y R  S3 Y s3 Y R  S3 - s4 - R  INITIAL\_STATE: s0  BLANK\_CHARACTER: -  FINAL\_STATES: s4 |

**5.2 Input String File**

The Turing Machine input string file is a text file with an “.str” extension. For the string to be valid the input string file must pass four checks. First each line of the input string file is an input string composed of characters from Σ. Second λ (empty string) is specified in the file as \ and must appear on its own line. Third duplicated strings are not allowed and will be discarded. Four if any other line is empty or contains a character not in Σ, the line will be discarded. If any of the checks fail the simulator will display an appropriate error message per the check. Please refer to fig 4-4 to find examples of valid and invalid strings.

|  |
| --- |
| Fig 4-4 Input String Example |
| Σ = { a, b, c }  aaaabacbcabcacbacba Valid input string  acbcde Invalid, { d, e } not in Σ  aaacbc\abc Invalid, \ must appear in its own line  \\ Invalid, { \ } not in Σ  \ Valid empty string |

There are no limits to the length of an input string, on the number of input strings or the number of strings. If the input string read from the file is valid, then it is stored in the string list.

1. **REFERENCES**

Dr. Neil B. Corrigan, Turing Machine Definition File and Input String File Example Handout

Dr. Neil B. Corrigan, Turing Machine Specification Handout

Emmanuel Bonilla, Turing Machine Application Requirements Specification