Software Test Plan

1.0 Test Plan Identifier

Turing Machine (TM) in C# 1.3.

1.11 Team 2 (Lansdon Page, Jason Wong, Ryan Wilson, Jason Stidham)

2.0 Testing Approach

2.1 Unit Testing and Coverage

We will be performing a series of unit tests using NUnit test suite verifying validity of the Parse function of the Turing Machine. In addition code coverage will identify the percentage of the application our tests will address.

2.2 Combinational Testing

We will be performing combinational testing on the run command using a black box approach. The Turing Machine has two variables we are allowed to manipulate using a manual approach, s(E)t and (T)runcate. In the combinational test table, figure 1, the values T and F represent True and False respectively. True represents a valid entry from the command line for the corresponding variable. Likewise False refers to an invalid entry made from the command line.

2.3 UML Test Cases

We are identifying test cases for the Turing machine from the UML Diagram; see figure 2, and using the aggregations from each of the necessary Turing machine objects as focal points for our tests.

3.0 Unit Test

The following pages detail the unit tests run, the results we derived from those unit tests, and the coverage that those tests achieved.

3.1 Unit Tests

Test ID	3.1.1 - ParseDefinition_CheckForDuplicateStates
Test Description	Verify Conformance of Requirement 4.2.1.4a of Requirements Document that State names must be unique. This is accomplished by feeding a definition file with two duplicate states to the load method of the States Class, and verifying a false boolean is returned.
Component	Type: Class method Name: States.load() bool States.load(List<string> definition_file)
Input Condition	List<string> invalid_definition = {“S1 S1”, “INPUT_ALPHABET:”} States test_state = new States()
Input State	Turing Machine program has just finished processing both the Turing Machine description and the keyword “STATES:”
Expected Results	A false boolean should be returned from the load class method of the States class.

Test ID	3.1.2 - ParseDefinition_CheckCaseSensitivityForStates
Test Description	Verify Conformance of Requirement 4.2.1.4b of Requirements Document that State names must be case sensitive. This is accomplished by feeding a definition file with two similarly named, but differingly cased states to the load method of the States class, and verifying that case is maintained by States class.
Component	Type: Class method Name: States.load() bool States.load(List<string> definition_file)
Input Condition	List<string> invalid_definition = {“state1 STATE1”, “INPUT_ALPHABET:”} States test_state = new States()
Input State	Turing Machine program has just finished processing both the Turing Machine description and the keyword “STATES:”
Expected Results	State class should contain two states, and maintain case sensitivity.

3.1 Unit Tests (Continued)

Test ID	3.1.3 - ParseDefinition_CheckForAtLeastOneState
Test Description	Verify Conformance of Requirement 4.2.1.2 of Requirements Document that there must be at least one state. This is accomplished by feeding a definition file with no states, and verifying that an exception was thrown.
Component	Type: Class method Name: States.load() bool States.load(List<string> definition_file)
Input Condition	List<string> invalid_definition = (“”, “INPUT_ALPHABET:”) States test_state = new States()
Input State	Turing Machine program has just finished processing both the Turing Machine description and the keyword “STATES:”
Expected Results	A false boolean should be returned from the load class method of the States class.

Test ID	3.1.4 - ParseDefinition_CheckForValidStateCharacters
Test Description	Verify Conformance of Requirement 4.2.1.4c of Requirements Document that only alphanumeric characters and the underscore character are allowed to be used in the naming of states. This is accomplished by feeding an invalid definition file using invalid characters, and verifying that an exception was thrown.
Component	Type: Class method Name: States.load() bool States.load(List<string> definition_file)
Input Condition	List<string> invalid_definition = (“$”, “INPUT_ALPHABET:”) States test_state = new States()
Input State	Turing Machine program has just finished processing both the Turing Machine description and the keyword “STATES:”
Expected Results	A false boolean should be returned from the load method of the States class.

3.1 Unit Tests (Continued)

Test ID	3.1.5 - ParseDefinition_CheckThatElementsAreLengthOne
Test Description	Verify Conformance of Requirement 4.2.2.2 of Requirements Document that input alphabet consists of elements of only length one. This is accomplished by feeding an invalid definition file that contains input alphabet elements of length greater than one.
Component	Type: Class Method Name: InputAlphabet.load() bool InputAlphabet.load(List<string> definition_file)
Input Condition	List<string> invalid_definition = ("ab", "TAPE_ALPHABET:") InputAlphabet test_inputalphabet = new InputAlphabet()
Input State	Turing Machine program has just finished processing both the Turing Machine description, the keyword "STATES:" and its elements, and the keyword "INPUT_ALPHABET".
Expected Results	A false boolean should be returned from the load class method of the InputAlphabet class.

Test ID	3.1.6 - ParseDefinition_CheckForDuplicateInputAlphabetCharacters
Test Description	Verify Conformance of Requirement 4.2.2.3 of Requirements Document that input alphabet consists of unique elements. This is accomplished by feeding an invalid definition file that contains duplicate input alphabet elements.
Component	Type: Class Method Name: InputAlphabet.load() bool InputAlphabet.load(List<string> definition_file)
Input Condition	List<string> invalid_definition = ("ab", "TAPE_ALPHABET:") InputAlphabet test_inputalphabet = new InputAlphabet()
Input State	Turing Machine program has just finished processing both the Turing Machine description, the keyword "STATES:" and its elements, and the keyword "INPUT_ALPHABET".
Expected Results	A false boolean should be returned from the load class method of the InputAlphabet class.

3.1 Unit Tests (Continued)

Test ID	3.1.7 - ParseDefinition_TransFunct_Valid
Test Description	Test that a transition function containing proper 5 components is parsed properly.
Component	transition_function
Input Condition	Definition input containing a transition function with 5 valid fields.
Input State	“TRANSITION_FUNCTION:” keyword has already been parsed.
Expected Results	all 5 fields are correctly parsed and stored in transition_function class.

Test ID	3.1.8 - ParseDefinition_TransFunct_InvalidFieldCount
Test Description	Test that a transition function NOT containing proper 5 components is produces an error.
Component	transition_function
Input Condition	Definition input containing a transition function without 5 valid fields.
Input State	“TRANSITION_FUNCTION:” keyword has already been parsed.
Expected Results	Should produce an error and return false.

Test ID	3.1.9 - ParseDefinition_TransFunct_InvalidChar
Test Description	Verify that accepted transitions have valid characters from the tape alphabet.
Component	transition_function
Input Condition	Definition containing a state that doesn’t exist in the tape alphabet
Input State	“TRANSITION_FUNCTION:” keyword has already been parsed.
Expected Results	Should have error

Test ID	3.1.10 - ParseDefinition_InitState_TooManyStates
Test Description	Validate only 1 initial state is accepted
Component	turing_machine
Input Condition	Give two initial states.
Input State	TRANSITION_FUNCTION: already parsed
Expected Results	Should have an error due to too many initial states.

3.1 Unit Tests (Continued)

Test ID	3.1.11 - ParseDefinition_BlankChar_NotInAlphabet
Test Description	Blank char must be member of tape_alphabet.
Component	turing_machine
Input Condition	blank char used that is not part of tape alphabet.
Input State	INITIAL_STATE: keyword has been parsed.
Expected Results	Should have an error.

Test ID	3.1.15 - ParseDefinition_FinalStates_NoStates
Test Description	Final states must have at least one final state.
Component	final_states
Input Condition	Pass 0 final states
Input State	INPUT_STATE: keyword has already been parsed.
Expected Results	Must have one or more final states

Test ID	3.1.16 - ParseDefinition_FinalStates_NotInStates
Test Description	Final State must be a member of states
Component	final_states
Input Condition	final_state defined that is not in states
Input State	FINAL_STATES: already parsed.
Expected Results	Should have error due to invalid final state.

3.2 Unit Test Results

Unique Test Identifier	3.1.1 - ParseDefinition_CheckForDuplicateStates
Result	Test Failed - False was returned by method.

Unique Test Identifier	3.1.2 - ParseDefinition_CheckCaseSensitivityForStates
Result	Test Passed - Indicating that the State class does maintain case.

Unique Test Identifier	3.1.3 - ParseDefinition_CheckForAtLeastOneState
Result	Test Failed - False was returned by the method.

Unique Test Identifier	3.1.4 - ParseDefinition_CheckForValidStateCharacters
Result	Test Failed - False was returned by the method.

Unique Test Identifier	3.1.5 - ParseDefinition_CheckThatElementsAreLengthOne
Result	Test Failed - False was returned by the method.

Unique Test Identifier	3.1.6 - ParseDefinition_CheckForDuplicateInputAlphabetCharacters
Result	Test Failed - Class method failed to detect duplicate Input Alphabet characters.

Unique Test Identifier	3.1.10 - ParseDefinition_TransFunct_Valid
Result	PASS

3.2 Unit Test Results (Continued)

Unique Test Identifier	3.1.11 - ParseDefinition_TransFunct_InvalidFieldCount
Result	FAILED - Invalid transition accepted.

Unique Test Identifier	3.1.12 - ParseDefinition_TransFunct_InvalidChar
Result	FAILED - Invalid char accepted.

Unique Test Identifier	3.1.13 - ParseDefinition_InitState_TooManyStates
Result	PASS

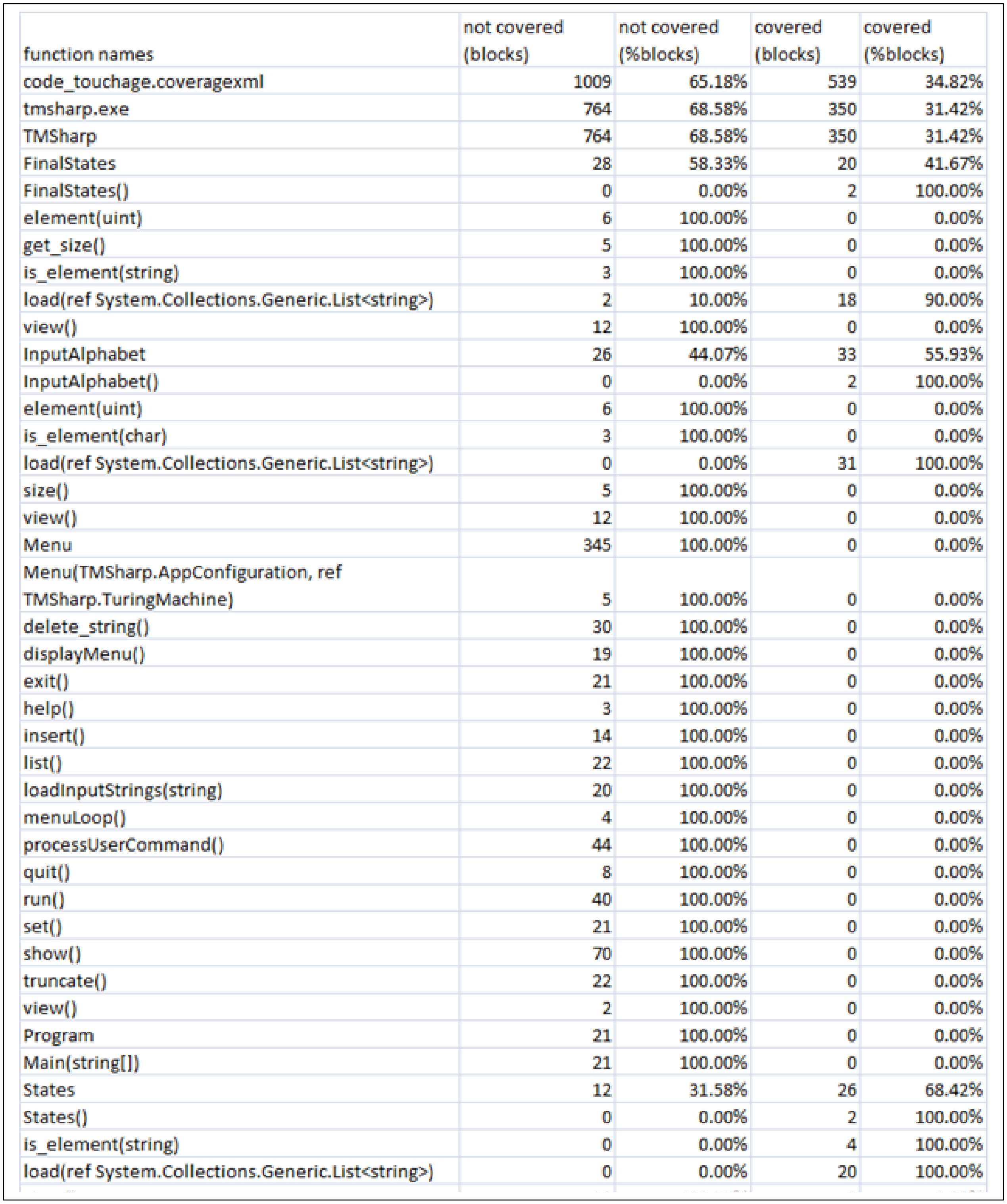
Unique Test Identifier	3.1.14 - ParseDefinition_BlankChar_NotInAlphabet
Result	FAIL - Invalid blank char accepted.

Unique Test Identifier	3.1.15 - ParseDefinition_FinalStates_NoStates
Result	PASS

Unique Test Identifier	3.1.16 - ParseDefinition_FinalStates_NotInStates
Result	FAIL - Invalid final state accepted.

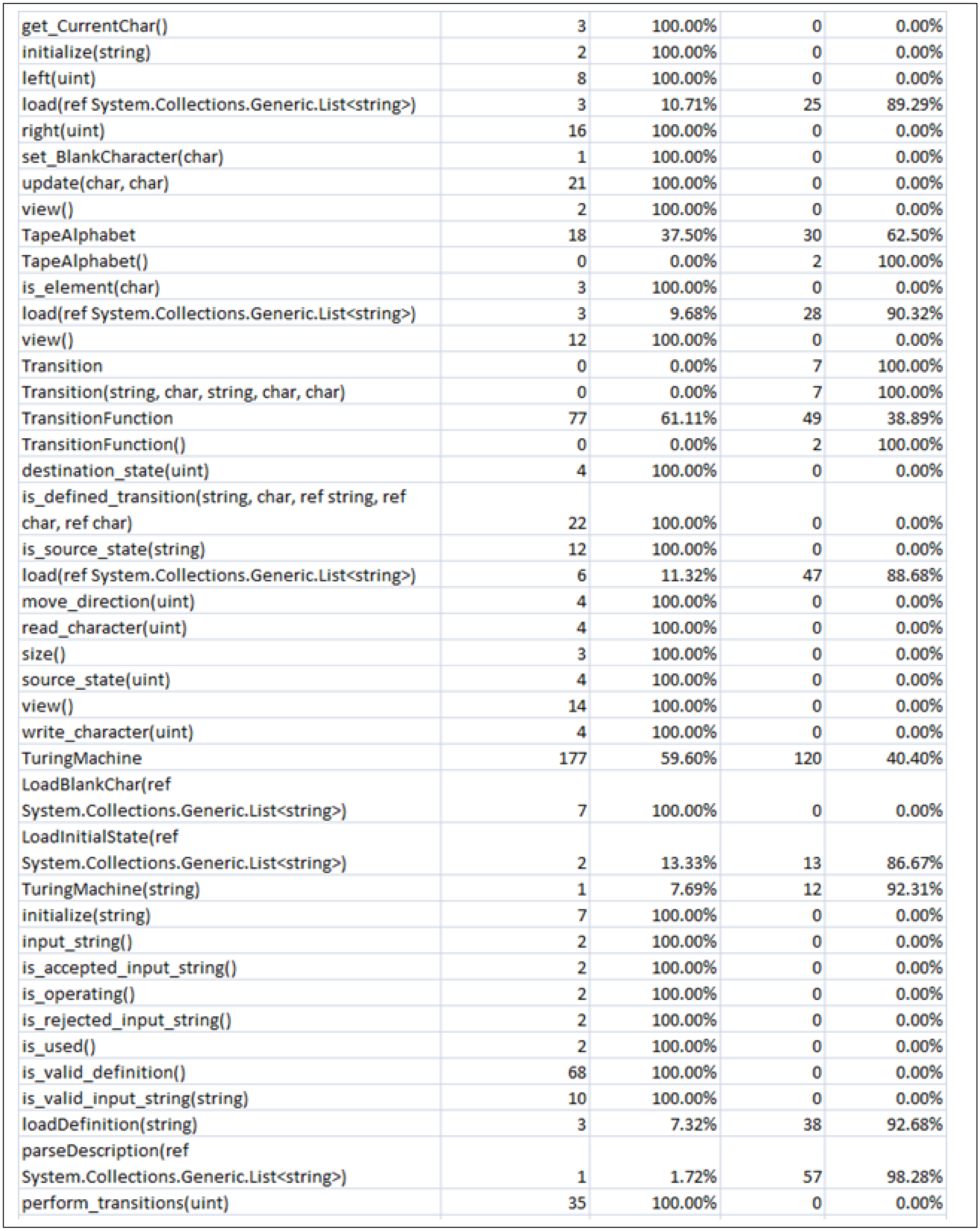
3.3 Unit Test Coverage

Using NUnit as our testing frame work we were able to provide 34.68% code coverage for the Turing Machine application. The chart below provides detailed information on the methods covered.



function names	not covered (blocks)	not covered (%blocks)	covered (blocks)	covered (%blocks)
code_touchage.coveragexml	1009	65.18%	539	34.82%
tmsharp.exe	764	68.58%	350	31.42%
TMSharp	764	68.58%	350	31.42%
FinalStates	28	58.33%	20	41.67%
FinalStates()	0	0.00%	2	100.00%
element(uint)	6	100.00%	0	0.00%
get_size()	5	100.00%	0	0.00%
is_element(string)	3	100.00%	0	0.00%
load(ref System.Collections.Generic.List<string>)	2	10.00%	18	90.00%
view()	12	100.00%	0	0.00%
InputAlphabet	26	44.07%	33	55.93%
InputAlphabet()	0	0.00%	2	100.00%
element(uint)	6	100.00%	0	0.00%
is_element(char)	3	100.00%	0	0.00%
load(ref System.Collections.Generic.List<string>)	0	0.00%	31	100.00%
size()	5	100.00%	0	0.00%
view()	12	100.00%	0	0.00%
Menu	345	100.00%	0	0.00%
Menu(TMSharp.AppConfiguration, ref TMSharp.TuringMachine)	5	100.00%	0	0.00%
delete_string()	30	100.00%	0	0.00%
displayMenu()	19	100.00%	0	0.00%
exit()	21	100.00%	0	0.00%
help()	3	100.00%	0	0.00%
insert()	14	100.00%	0	0.00%
list()	22	100.00%	0	0.00%
loadInputStrings(string)	20	100.00%	0	0.00%
menuLoop()	4	100.00%	0	0.00%
processUserCommand()	44	100.00%	0	0.00%
quit()	8	100.00%	0	0.00%
run()	40	100.00%	0	0.00%
set()	21	100.00%	0	0.00%
show()	70	100.00%	0	0.00%
truncate()	22	100.00%	0	0.00%
view()	2	100.00%	0	0.00%
Program	21	100.00%	0	0.00%
Main(string[])	21	100.00%	0	0.00%
States	12	31.58%	26	68.42%
States()	0	0.00%	2	100.00%
is_element(string)	0	0.00%	4	100.00%
load(ref System.Collections.Generic.List<string>)	0	0.00%	20	100.00%

3.3 Unit Test Coverage (Continued)



get_CurrentChar()	3	100.00%	0	0.00%
initialize(string)	2	100.00%	0	0.00%
left(uint)	8	100.00%	0	0.00%
load(ref System.Collections.Generic.List<string>)	3	10.71%	25	89.29%
right(uint)	16	100.00%	0	0.00%
set_BlankCharacter(char)	1	100.00%	0	0.00%
update(char, char)	21	100.00%	0	0.00%
view()	2	100.00%	0	0.00%
TapeAlphabet	18	37.50%	30	62.50%
TapeAlphabet()	0	0.00%	2	100.00%
is_element(char)	3	100.00%	0	0.00%
load(ref System.Collections.Generic.List<string>)	3	9.68%	28	90.32%
view()	12	100.00%	0	0.00%
Transition	0	0.00%	7	100.00%
Transition(string, char, string, char, char)	0	0.00%	7	100.00%
TransitionFunction	77	61.11%	49	38.89%
TransitionFunction()	0	0.00%	2	100.00%
destination_state(uint)	4	100.00%	0	0.00%
is_defined_transition(string, char, ref string, ref char, ref char)	22	100.00%	0	0.00%
is_source_state(string)	12	100.00%	0	0.00%
load(ref System.Collections.Generic.List<string>)	6	11.32%	47	88.68%
move_direction(uint)	4	100.00%	0	0.00%
read_character(uint)	4	100.00%	0	0.00%
size()	3	100.00%	0	0.00%
source_state(uint)	4	100.00%	0	0.00%
view()	14	100.00%	0	0.00%
write_character(uint)	4	100.00%	0	0.00%
TuringMachine	177	59.60%	120	40.40%
LoadBlankChar(ref System.Collections.Generic.List<string>)	7	100.00%	0	0.00%
LoadInitialState(ref System.Collections.Generic.List<string>)	2	13.33%	13	86.67%
TuringMachine(string)	1	7.69%	12	92.31%
initialize(string)	7	100.00%	0	0.00%
input_string()	2	100.00%	0	0.00%
is_accepted_input_string()	2	100.00%	0	0.00%
is_operating()	2	100.00%	0	0.00%
is_rejected_input_string()	2	100.00%	0	0.00%
is_used()	2	100.00%	0	0.00%
is_valid_definition()	68	100.00%	0	0.00%
is_valid_input_string(string)	10	100.00%	0	0.00%
loadDefinition(string)	3	7.32%	38	92.68%
parseDescription(ref System.Collections.Generic.List<string>)	1	1.72%	57	98.28%
perform_transitions(uint)	35	100.00%	0	0.00%

4.0 Combinational Tests

For our combinational testing, it was decided that the decision table would be the most appropriate test model because it served as our list of test cases as well as our test results. It allowed for a clear representation of the variables to be tested against, the inputs used for said variables, the results we expected, and the actual results we received –all in one concise table.

4.1 Combinational Test Models

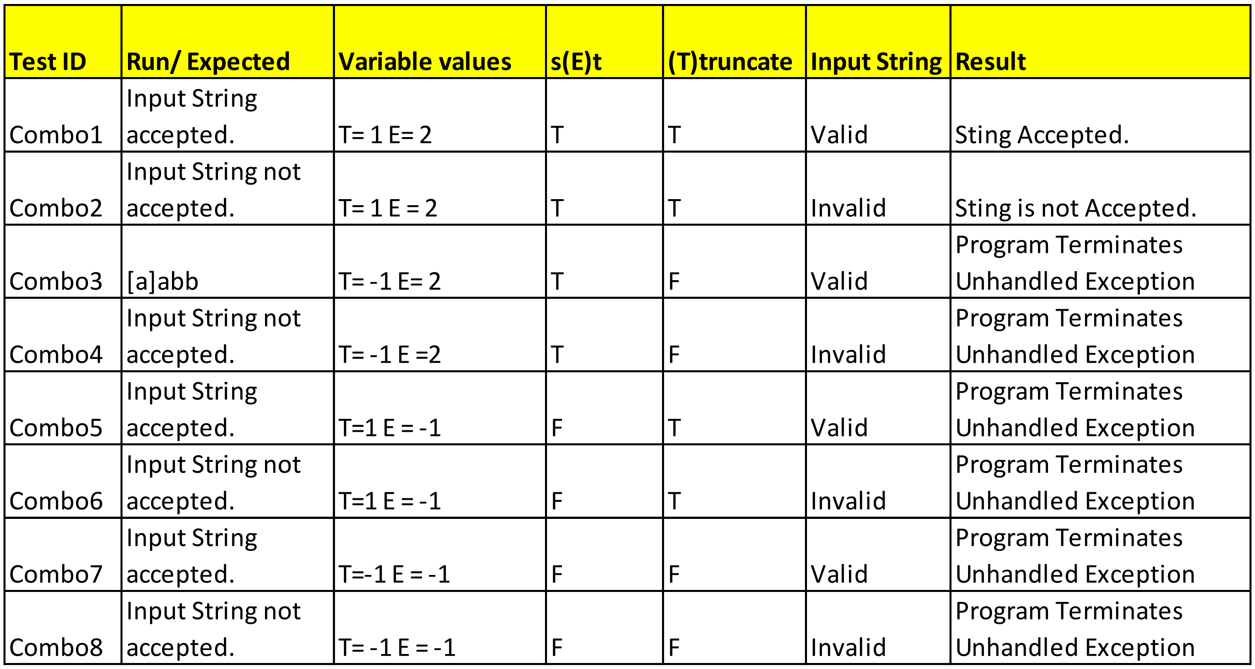
Combination tests will be ran using a manual black box approach. The Turing Machine has two variables we are allowed to manipulate using a manual approach, s(E)t and (T)runcate.

4.2 Combinational Test Case

Reference columns Run/Expected in figure one for a description of the test to be conducted for the corresponding TestID.

4.3 Combinational Test Results

Reference column Result in figure one for a description of the test to be conducted for the corresponding TestID.



Test ID	Run/ Expected	Variable values	s(E)t	(T)truncate	Input String	Result
Combo1	Input String accepted.	T= 1 E= 2	T	T	Valid	Sting Accepted.
Combo2	Input String not accepted.	T= 1 E = 2	T	T	Invalid	Sting is not Accepted.
Combo3	[a]abb	T= -1 E= 2	T	F	Valid	Program Terminates Unhandled Exception
Combo4	Input String not accepted.	T= -1 E =2	T	F	Invalid	Program Terminates Unhandled Exception
Combo5	Input String accepted.	T=1 E = -1	F	T	Valid	Program Terminates Unhandled Exception
Combo6	Input String not accepted.	T=1 E = -1	F	T	Invalid	Program Terminates Unhandled Exception
Combo7	Input String accepted.	T=-1 E = -1	F	F	Valid	Program Terminates Unhandled Exception
Combo8	Input String not accepted.	T= -1 E = -1	F	F	Invalid	Program Terminates Unhandled Exception

Figure 1

5.0 UML Diagram

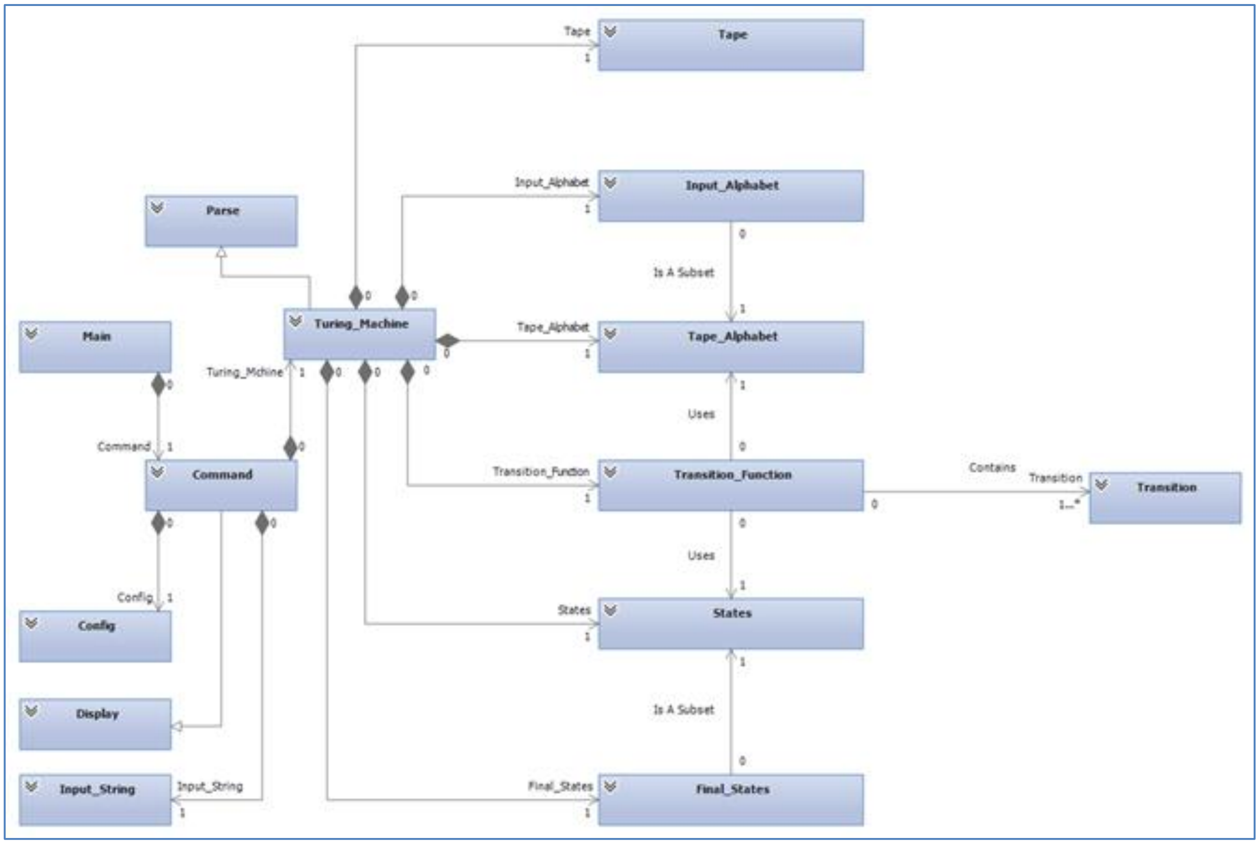


Figure 2

5.1 UML Test Case

Test ID:	UML tc1 - Check for Tape Object
Description:	Verifies that a Tape object is instantiated for the Turing Machine.
Expected Result:	Upon instantiation of the Turing machine the object Tape will be generated.

Test ID:	UML tc2 - Check for Input Alphabet Object
Description:	Verifies that a Input_Aphabet is instantiated for the Turing Machine.
Expected Result:	Upon instantiation of the Turing machine the object Input_Alphabet will be generated.

Test ID:	UML tc3 - Check for Tape Alphabet Object
Description:	Verifies that a Tape_Alphabet object is instantiated for the Turing Machine.
Expected Result:	Upon instantiation of the Turing machine the object Tape_Alphabet will be generated.

Test ID:	UML tc4 - Check for Final States Object
Description:	Verifies that a Transition_Function object is instantiated for the Turing Machine.
Expected Result:	Upon instantiation of the Turing machine the object Transition_Function will be generated.

Test ID:	UML tc5 - Check for States Object Instantiation
Description:	Verifies that a States object is instantiated the Turing Machine.
Expected Result:	Upon instantiation of the Turing machine, the object States will be generated.

Test ID:	UML tc6 - Check for Final States Object
Description:	Verifies that the Final_States object is an instance of the Turing Machine.
Expected Result:	Upon instantiation of the Turing machine the objectFinal_ States will be generated.

5.1 UML Test Case (continued)

Test ID:	UML tc7 - Check for Tape Alphabet Subset
Description:	Verifies that the Transition object is an instance of the Turning Machine.
Expected Result:	Upon instantiation of the Turing machine the object Transition will be generated.

Test ID:	UML tc8 - Check the Tape Alphabet Subset
Description:	Verifies that Input_alphabet is a subset of tape_alphabet
Expected Result:	The content of the input alphabet is infact a subset of the tape alphabet.

Test ID:	UML tc9 - Check the Final States Subset
Description:	Verifies that each final state is an element of the set of States.
Expected Result:	All elements of the set Final States are also elements of the set States.