CS 410 Automata Theory and Formal Languages

Project 1 Design Report

Bora Özügüzel

S014465

1. Introduction

In this homework, a project that reads a text file of a Non-Deterministic Automata. The project, then coverts it to a Deterministic Automata. The result then printed to the console in the same format. The epsilon transitions are ignored. The text file has following partitions: ALPHABET, STATES, START, FINAL, TRANSITIONS. The file ends with END.

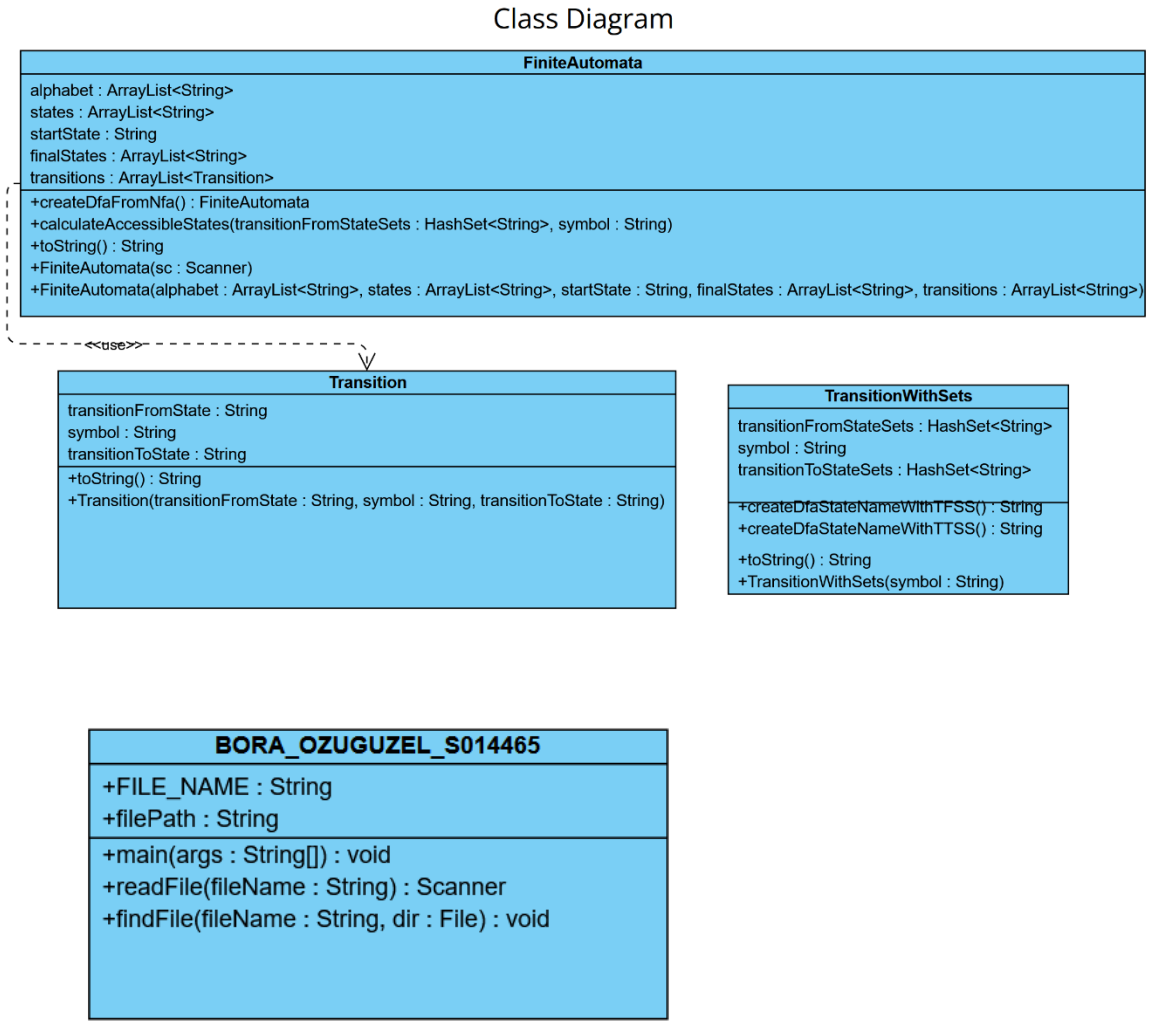
File Format:

ALPHABET
symbol1
symbol2
…
symbolZ
STATES
state1
state2
…
stateZ
START
startState
FINAL
finalState1
finalState2
…
finalStateN
TRANSITIONS
state1 symbol1 stateZ
state1 symbol2 stateY
….
state1 symbolZ stateX
…
stateZ symbolY stateK

END
Names of given files used in this project:
NFA1.txt
NFA2.txt
The language chosen for this project is Java.

2. Classes and Methods

Class Diagram of the project is given below:



3. Implementation Details

In the main method a method to read the NFA1.txt file will be called named readFile. The method tries to find the file in the project folder with the given filename defined in FILE_NAME for example NFA1.txt. If it cannot find it directly, it will try to search

the file in the whole project with the method names findFile. This file will be accessed by the Scanner class from the Java.Util Package. The result will be send to instantiate a new FiniteAutomata object, which can hold the information of both Non-Deterministic Finite Automata and Deterministic Finite Automata. The constructor shown as below will be called:

public FiniteAutomata(Scanner sc) {…}

In a while loop the scanned filed will be parsed line by line.
While constructing the object, an integer called partitionState is kept holding which partition between the titles in the txt file in. If the partitionState is 1, which means that it is currently reading the lines between ALPHABET and STATES. It will add every line to a string ArrayList called alphabet. The same will happen for states and final states. There is only one start state and it is kept in a string called startState. For transitions an ArrayList of a new object called Transition. In the file for transitions three strings with a space between them is given. The whole line is tokenized with StringTokenizer from Java.Util.
The values are transitionFromState, symbol and transitionToState, where transitionFromState represents which state does the edge start from. For symbol, which is the value that is from the alphabet and for the transitionToState which State does the edge connected to. After this all value in the NFA1.txt file is assigned to the FiniteAutomata Object instantiated with the name nfa.

FiniteAutomata has a method called createDfaFromNfa as shown below:

public FiniteAutomata createDfaFromNfa() {…}

A new object called TransitionWithSets is needed to hold more complex information. TransitionWithSets is similar to the Transition object, since they both have the symbol attribute, but for other values it is different. A HashSet called transitionFromStateSets and transitionToStateSets are used. HashSets can contain same value only once, so every state that will be stored in them will be distinct. This class shows which states can be accessed with the symbol value from the alphabet from the states in the transtionFromStateSets HashSet.

A List for the new object called TransitionWithSets is created called transitionWithSetsList. In a for loop for each symbol of the alphabet, the start state is converted to a HashSet, then it is sent to a method called calculateAccessibleStates with the symbol value. The result is then added to the transitionWithSetsList even if there are no accessible states. This converts Transition objects of the start state to TransitionWithSets even if it is accessing the empty state.

The method calculateAccessibleStates creates a new TransitionWithSets object by calculating which states are accessible to every state given in transitionFromStateSets with the symbol given. The parameters of the method are shown as below:

public TransitionWithSets calculateAccessibleStates(HashSet<String> transitionFromStateSets, String symbol) {…}

The set of accessible states are created in a lambda expression using the stream method from Java.Util. In a for loop, for each state in transitionFromStateSets, the state is searched in the transitions ArrayList where their symbols are same. The resulting Transition object's transtionToState string are the accessible states.

In a while loop a List of HashSet of strings are created with a stream lambda expression. All the transitionToStateSets from the transitionWithSetsList that aren't found in the transitionFromStateSets are formed to be a list called newStateSets. If empty set exists, even the empty set is added. In a for loop each of the HashSets, there is another for loop for each symbols of alphabet. In these loops, the accessible states are calculated using the hashSets and the symbol with the calculateAccessibleStates method. Then, the TransitionWithSets object added to the transitionWithSetsList list, so they can be used by the stream lambda expression in at the next iteration. This while loop continues until stream lambda expression return nothing.

The DFA will use the same alphabet so the alphabet of the NFA is cloned. The states of DFA are named after the sets in transitionWithSetslist. All distinct transitionFromStateSets of the transitionWithSetList are converted to special names using the elements of the set with createDfaStateNameWithTFSS. When using a HashSet which contains A state, q{A} is created. For an empty set, q{} is created. If there are multiple states like A and B, q{A, B} is created. The start state of the DFA is created in the same way. It is created from the start state of the NFA. The final states are calculated with a stream lambda expression. In the transitionWithSetslist, all transitionToStateSets that contain any one of the final states of the NFA are final states of the DFA. These are converted to state name with createDfaStateNameWithTTSS. Transitions of the DFA is calculated by converting transitionWithSetslist to a ArrayList of Transition objects from the List of TransitionWithSets with a stream lambda expression. The methods, createDfaStateNameWithTFSS and createDfaStateNameWithTTSS are used to convert sets to state names and the symbol stays the same. The results are created as temporary ArrayLists, they are used to create a new FiniteAutomata object with the constructor below at the end of the method:

return new FiniteAutomata(tempAlphabet, tempStates, tempStartState , tempFinalStates, tempTransitions);

ToString methods of these classes are overridden, so when printing the new dfa object, the result will be printed.

4. Results

NFA1.txt

NFA1 input from file:

ALPHABET
0
1
STATES
A
B
C
START
A
FINAL
C
TRANSITIONS
A 0 A
A 1 A
A 1 B
B 1 C
END

DFA1 output to console:

ALPHABET
0
1
STATES
q{A}
q{A, B}
q{A, B, C}
START
q{A}
FINAL
q{A, B, C}
TRANSITIONS
q{A} 0 q{A}
q{A} 1 q{A, B}
q{A, B} 0 q{A}
q{A, B} 1 q{A, B, C}
q{A, B, C} 0 q{A}
q{A, B, C} 1 q{A, B, C}
END

NFA2.txt

NFA2 input from file:

ALPHABET
0
1
STATES
A
B
C
START
A
FINAL
C
TRANSITIONS
A 0 A
A 1 B
A 1 C
B 0 B
B 0 C
C 0 A
C 0 B
C 1 B
END

DFA2 output to console:

ALPHABET
0
1
STATES
q{A}
q{B, C}
q{A, B, C}
q{B}
q{}
START
q{A}
FINAL
q{B, C}
q{A, B, C}
TRANSITIONS
q{A} 0 q{A}
q{A} 1 q{B, C}
q{B, C} 0 q{A, B, C}
q{B, C} 1 q{B}

q{A, B, C} 0 q{A, B, C}
q{A, B, C} 1 q{B, C}
q{B} 0 q{B, C}
q{B} 1 q{}
q{} 0 q{}
q{} 1 q{}
END