Final Automata Project: Luis Alcantar

This report contains information about how a language can be translated into a Finite Automata. The structure used to implement the NFA is a type of graph which is made out of two classes; States and Transition. The states represent vertices and Transition represents edges. There are two important methods in the process; the first being the method factoryR that parse the string to create the non-deterministic finite automata and takes advantage or recursion. The second method makeDfa is used for the deterministic finite automata that use different data structures such as Maps, HashSets, ArrayLists.

All the specification can be found in the description below or the code itself.

The following classes were used:

Automata

This class makes an automaton. The fields are:

- ArrayList<State> states: map containing all the states of the Automaton
- State startState: Pointer to the start state of the Automaton

State

This class simulates a vertex in a graph. The fields and methods are:

- Static int nameID: State names will be an integer. This variable is used to represent that number incrementing every time a new State is created.
- Int stateName: the name of the state, controlled by nameID.
- List<Transition> transitions: map with all the transitions of the state Character which represent the transition to another state, as well as the pointer to the destination state.
- Boolean isStartState: to check if state is start or not.
- Boolean isFinalState: to check if state is final or not.

- Public Boolean isStartState(): this method returns true or false, depending if the state is start or not.

- Public Boolean isFinalState(): this method returns true or false, depending if the state is final or not.

- Public static int assignNameID(): get the name ID to assign to a new state and increment for the next call.

Transition

- State pointTo: the destination state of the transition
- char transition: the character with which it goes to said state

- void addTransition(State s, char t): takes a state and a character and makes a new transition with it.
- State getTransitionName(): returns the name of this transition.

AutomataFactory

This class builds the automaton and implements all the different operations (concatenation, union, star, etc.) The fields and methods are:

- Static Set<Character> setAlpha: it is the Alphabet of the Automaton.

- Static Automata automata: this is used as a temporal variable to hold the working automata, as well as a holder for the final product before being returned.

- Static Automata dfa: this is used as a temporal variable to hold the working DFA automata.

- Void makedfa(String Language): Creates a DFA with the given NFA. It uses the NFA to generate sets of epsilon closures to find the transition of the DFA.The result is store in the global variable dfa.

- Void checkStringValid(String L,State s, Int index) : Takes s string and the start state of the automata that is testing. With the start state will traverse the machine and find if is recognized or not by the machine.

- Set<State> fidSets(): this methods is use by the makedfa() method to find all final methods of the nfa.This facilitate to compare which methods in the DFA are final and which ones not.

- String findParenthesis(String s, int i): as soon as an opening parenthesis is found, this finds the closing parenthesis corresponding to the position i and returns that substring.

- Automata factoryR(String s, int x): This method is use to create the nfa,does a recursive call for each nested parenthesis found, and processes that inner automaton before going on to the outer one, building the Automaton from the inside out, keeping the order of operations. It also calls makeStar, makePlus and so on. For single Characters, it creates a simple automaton that then is pushed on to the main stack.

- Int symbolValue(char ch): this method uses a switch statement that returns different values depending on the operation symbol, used in factoryR

- Void getAlpha(String s): Get the alphabet of the language using a Set to store all the characters of the String.

- Automata createSimpleAutomata(char ch): this method takes a character and makes a
- Automata unionAutomatas(Automata Automata1, Automata Automata2): Create a new automta by createing a new state , it connects the new state to the start state of both automata1 and automata2. Also the start state of automata1 and automata 2 are the new state created.

- Automata concatinate(Automata Automata1, Automata Automata2):Create a new atomata by conecting all the final states of automata1(parameter) to the start state of the autaomta2(parameter).and returns the new automata

- Automata makePlus(Automata temp): This method takes a Automata and will make it plus, each final transition in the automata will be connected to the start state of the automata.
- Automata makeStar(Automata temp):] Creates a new Automata. This automata will use the parameter temp and make it start using the method plus and crating a new final state that will have a transition to the start state of the automata temp.

- Void print(Automata temp): Helper method to print all the states of the machine and the conections

- HashSet<State> addEpsilons (Satate s, Boolean isStart,char ch): the method takes a state and find if there is a transition that with the given character takes to a new state. If and only if the condition is satisfy the method also look for sets that can take further with epsilon transitions. At the end of the method it return a set containing the states that were reachable.

Test Cases Justification:

All the test cases can be found in folder Test. The reason that I choose to work with thee test cases is because I was able to test all the operators. Also Considering these cases I checked all cases were my program could fail. These cases prove it is possible to convert a language to nfa and to dfa.

Failure:

(1) Not able to implement minimization.
(2) Consider language (a.b) if we try to recognize the string “abbbb”, the program will show that is recognized because it loops the last state.
(3) Complement can only be done to the whole language furthermore i was no able to implement intersection.