CONVERTER NFA TO DFA IN JAVA

Prepared by Musa Berkay Kocabaşoğlu S018431

1- Introduction

DFA refers to Deterministic Finite Automaton and NFA refers to Nondeterministic Finite Automaton. A Finite Automata (FA) is said to be deterministic if corresponding to an input symbol, there is a single resultant state i.e. there is only one transition but is said to be non-deterministic if there is more than one possible transition from one state on the same input symbol.

I implement a Java program that converts NFAs to DFAs in my project. NFA can use Empty String transition. However, I don't use empty string transition which is epsilon transition in our project because I ignore this transition in this assignment. NFA and DFA is set of five tuples represented as,

M = {Q, Σ, δ, qo, F}

Q: A non-empty finite set of states. (qo, q1, q2, …)
Σ: A non-empty finite set of input symbols. (0, 1, ...)
δ: It is a transition function that takes two arguments, a state, and an input symbol
qo: It is starting state, one of the states in Q.
F: It is a non-empty set of final states/ accepting states from the set belonging to Q.

2- Method

Firstly, I read the NFA txt file with File and Scanner built-in functions. I load alphabet, states, start state, final states, transitions of NFA to different data structures. I will explain which data structures I use in implementation part. After loading data to my program, I use this algorithm:

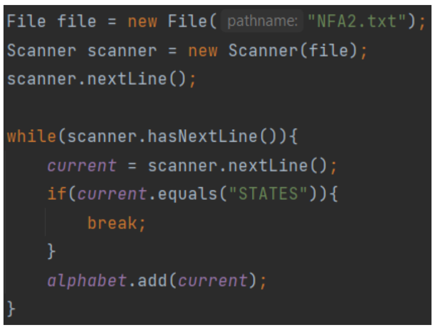
Let, M = (Q, ∑, δ, q0, F) is an NFA which accepts the language L(M). There should be equivalent DFA denoted by M' = (Q', ∑', δ', q0', F') such that L(M) = L(M'). I implement these steps:

1- Initially, I define Q' is ϕ.
2- After, I add start state of NFA to Q'.
3- Then I find the transitions from this start state.
4- I find the possible set of states for each input symbol.
5- If this set of states is not in Q', I add it to Q'.
6- If there is many target states, I write these states with one space in data structure. Ex: A 1 A B
7- After the traversing all transitions, I construct new DFA with NFA.

Finally, I print new DFA with for loops. Since alphabet and start state have not changed, I print alphabet and start state I used at the beginning. However, since new states are added and it is necessary to print transitions, I delete the spaces I added at the converting part. Moreover, since my dead end states are empty, I also check them and write with conditions. When printing the final state, I also use the union of symbols containing the final state set of the NFA in the DFA.

3- Implementation

In the beginning, I will import these built-in functions which are java.io.File, java.util.Scanner, java.io.FileNotFoundException for file loading, handling exceptions, and java.util.HashMap, java.util.ArrayList for necessary data structures. I read the file in load() function and I load the data step by step with while loop and scanner. I use condition for stopping while loop and pass loading a new variable. For doing this, I use String variable named current.



```
File file = new File( pathname: "NFA2.txt");
Scanner scanner = new Scanner(file);
scanner.nextLine();

while(scanner.hasNextLine()){
    current = scanner.nextLine();
    if(current.equals("STATES")){
        break;
    }
    alphabet.add(current);
}
```

I construct these variables with these type for loading data:

Alphabet → ArrayList<String>

States → ArrayList<String>

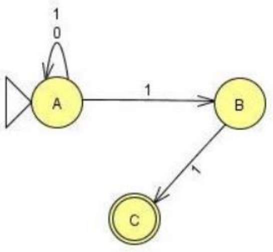
Start State → String

Final States → ArrayList<String>

NFA’s Transitions → HashMap<String, HashMap<String, String>>

DFA’s Transitions → HashMap<String, HashMap<String, String>>

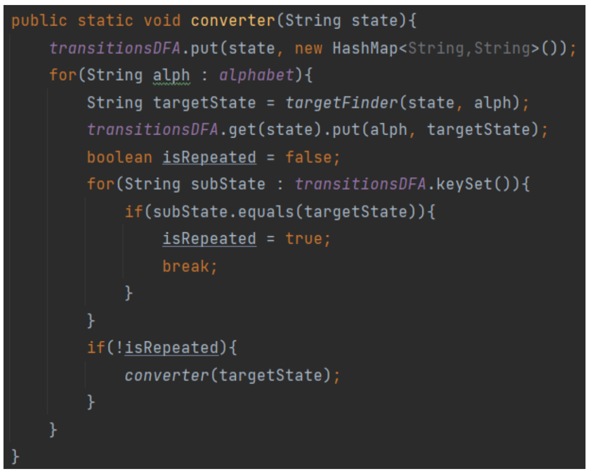
NFA’s transition structure is nested HashMap and look like this for right example:



{ “A”: {“0”: ”A”,

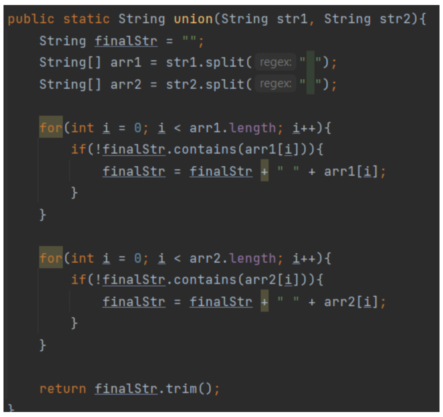
”1”: ”A B”},

”B”:{“1”: ”C”}}



```
public static void converter(String state){
    transitionsDFA.put(state, new HashMap<String,String>());
    for(String alph : alphabet){
        String targetState = targetFinder(state, alph);
        transitionsDFA.get(state).put(alph, targetState);
        boolean isRepeated = false;
        for(String subState : transitionsDFA.keySet()){
            if(subState.equals(targetState)){
                isRepeated = true;
                break;
            }
        }
        if(!isRepeated){
            converter(targetState);
        }
    }
}
```

After loading the data to our variables, I use the converter(String state) method to convert NFA transitions to DFA transitions. Firstly, I added start state to our DFA’s transitions variable of new key and create new HashMap<String, String> as value which is named transitionsDFA. Then, I look symbols in alphabet with for loop. I find the target state with targetFinder(String state, String alphabet) function. In targetFinder function, I find out which states the target state reaches with that symbol in nested for loop. When I reach the target states whether is single or multiple, I use union(String str1, String str2) function for combining target states. For example, if I have 2 target states A and B, I put these states like “A B” in the value of the value at transitionsDFA. I use trim method for finalStr variable because finalStr may remain empty or there are unnecessary gaps at the beginning and end. After that, I looked this targetState is as a key value in transitionDFA. If it is in, I don’t need to use converter for that targetState because it’s already filled in transitionsDFA but it’s not, I use converter function with this targetState. So, I can say that converter method is working recursively and fill transitionsDFA variable.

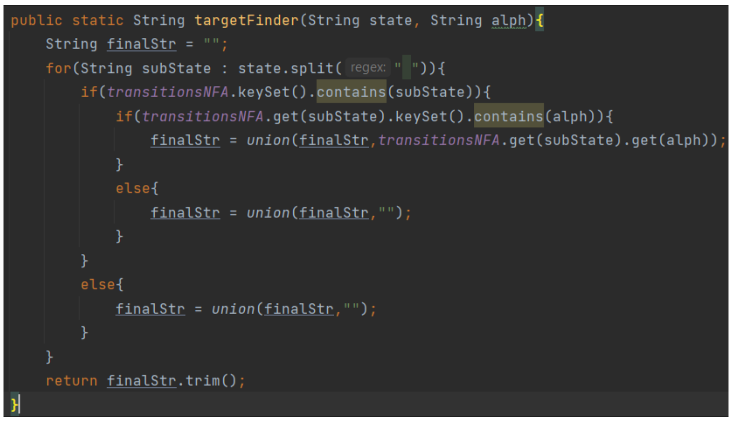


```
public static String union(String str1, String str2){
    String finalStr = "";
    String[] arr1 = str1.split( regex: " ");
    String[] arr2 = str2.split( regex: " ");

    for(int i = 0; i < arr1.length; i++){
        if(!finalStr.contains(arr1[i])){
            finalStr = finalStr + " " + arr1[i];
        }
    }

    for(int i = 0; i < arr2.length; i++){
        if(!finalStr.contains(arr2[i])){
            finalStr = finalStr + " " + arr2[i];
        }
    }

    return finalStr.trim();
}
```

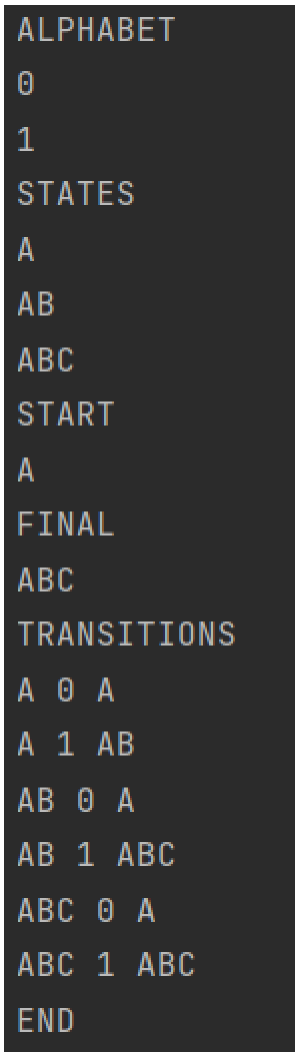


```
public static String targetFinder(String state, String alph){
    String finalStr = "";
    for(String subState : state.split( regex: " ")){
        if(transitionsNFA.keySet().contains(subState)){
            if(transitionsNFA.get(subState).keySet().contains(alph)){
                finalStr = union(finalStr,transitionsNFA.get(subState).get(alph));
            }
            else{
                finalStr = union(finalStr,"");
            }
        }
        else{
            finalStr = union(finalStr,"");
        }
    }
    return finalStr.trim();
}
```

Finally, I fill the transitions of DFA with converter. So, I use printDFA() function for printing our new alphabet, states, start state, final states and transitions in the same format with the given input text files. I use for loop for printing variables except start state because it is single value. Since alphabet and start state have not changed, I print alphabet and start state I used at the beginning. However, since new states are occured in DFA and it is necessary to print transitions, I delete the spaces I added at the converting part. Moreover, since my dead end states are implemented empty string like “” in transitionsDFA variable, I also check them and write with conditions.

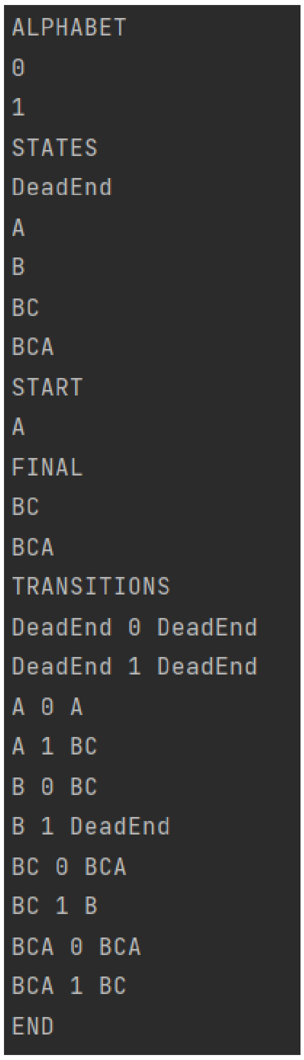
4- Results

The images below show the output produced after the run of the main method. This outputs show DFA1 and DFA2, the converted version of the NFAs in the NFA1 and NFA2 files.



```
ALPHABET
0
1
STATES
A
AB
ABC
START
A
FINAL
ABC
TRANSITIONS
A 0 A
A 1 AB
AB 0 A
AB 1 ABC
ABC 0 A
ABC 1 ABC
END
```

DFA1



```
ALPHABET
0
1
STATES
DeadEnd
A
B
BC
BCA
START
A
FINAL
BC
BCA
TRANSITIONS
DeadEnd 0 DeadEnd
DeadEnd 1 DeadEnd
A 0 A
A 1 BC
B 0 BC
B 1 DeadEnd
BC 0 BCA
BC 1 B
BCA 0 BCA
BCA 1 BC
END
```

DFA2