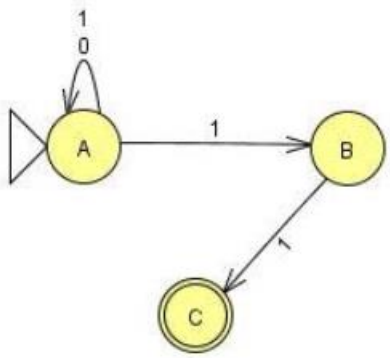
Introduction

The problem in this assignment is to create a software that takes a Non-deterministic Finite Automata (NFA) as an input and converts in into a Deterministic Finite Automata (DFA). The input is given as a text file with the format below:

```
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
```



To represent the NFA (right side), a text file (left side) is given. Software shall read the content of the file and write DFA with the same format in the console.

Method

An NFA can have multiple transitions from a specific state to other states including the source state itself. However, in a DFA, a state can have one and only one transition to another state or to itself for the each symbol of the alphabet.

Let's represent the NFA as $M = (Q, \sum, \delta, q0, F)$ where;

- Q: List of all states
- ∑: Alphabet
- δ: Transitions
- q0: Starting state
- F: Final state(s)

We want to create an equivalent DFA of this NFA that can be represented by $M' = (Q', \sum', q0', \delta', F')$.

The basic workflow of the algorithm that converts M to M' is;

1. Set $Q' = \phi$ as the initial list of states
2. Add q0 to Q' and find the transitions of this state

3. In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then add it to Q'.
4. In DFA, the F' (final state(s)) will be all the states that contain F

Implementation

Java is used with a procedural approach in the project's implementation part. To keep the NFA in a structured way in Java, a nested Hashmap is used. This nested Hashmap is used to check each possible state-symbol pair for creating the DFA.

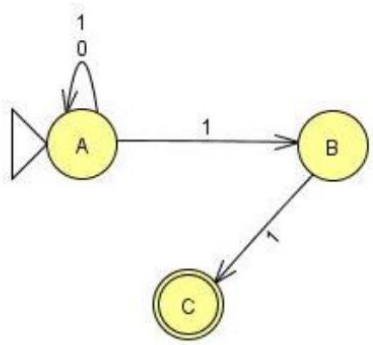


Figure 1: NFA Example

For the given NFA example in *Figure 1*, the nested Hashmap structure is

```
General = {"A": {"0": ["A"],
                 "1": ["A", "B"]},
           "B": {"0": ["Ø"],
                 "1": ["C"]},
           "C": {"0": ["Ø"],
                 "1": ["Ø"]}
          }
```

The program first reads the lines of the input file and stores the symbols to a separate list. After that, it creates the nested HashMap of the NFA by putting the states as the key values and then adds the transitions as the values. For keeping the final state(s), it uses an ArrayList<String> since there might be multiple final states.

When the forming of the NFA is done, it starts to create the DFA. At the beginning, DFA is a HashMap that has only the start state as the key with no values. At this point, I put the DFA in an iteration where its key values (states) are processed one by one. In this iteration, program checks the state's target states and if there are more than one target states, it creates a new state for this combination and adds it to DFA. To name the new state, it basically combines the names of the single states. If the list of target

states contain the final state of the NFA as well, it adds this new state to the final state list of the DFA. If there is only one target state and it is not put in the DFA before, it puts that single state to DFA with the values taken from the NFA. Program ensures that a state that is processed once is not processed again and this prevents possible bugs. At the end of the iteration process, DFA gets almost ready. At this point, the only need is to cover the empty state.

The program has only one function called “faNullConverter” that takes the DFA and puts “Ø” state if the DFA has an empty state transition from other states. In this case, “Ø” state connects to itself for all symbols. **(The “Ø” sign can be shown as “Ã?” if the program is run from the Windows command line! I did not fix it since “Ø” is the right sign to represent the empty set. It is shown correctly when I run it in VSCode)**

When the DFA is formed completely, program writes the components (alphabet, states, start state, final state(s), transitions) to the console with the same format of the input file.

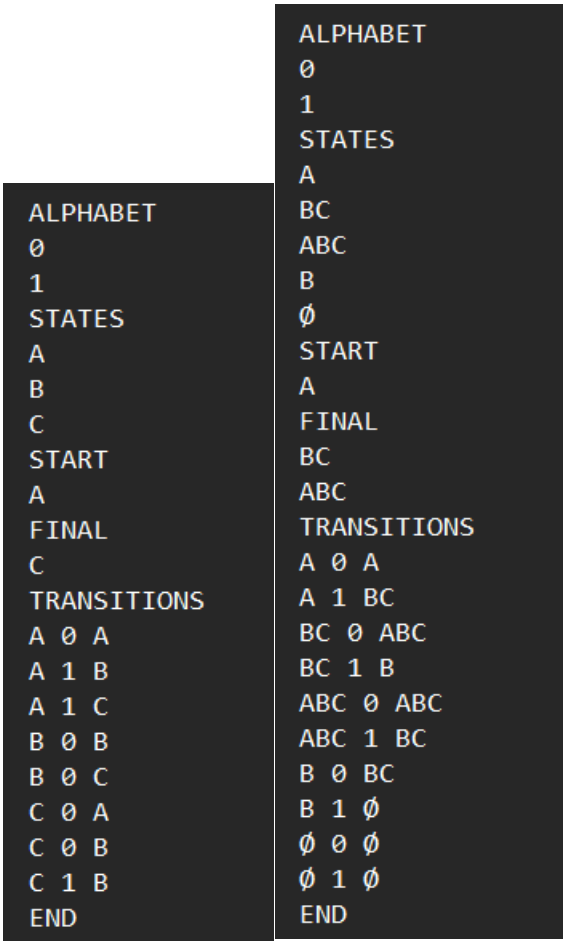
Results

Below shows the NFA1.txt (left side) content and the output DFA content (right side) created by the software.

```
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
```

```
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
```

Below shows the NFA2.txt (left side) content and the output DFA content (right side) created by the software.



```
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
```

```
ALPHABET
0
1
STATES
A
BC
ABC
B
Ø
START
A
FINAL
BC
ABC
TRANSITIONS
A 0 A
A 1 BC
BC 0 ABC
BC 1 B
ABC 0 ABC
ABC 1 BC
B 0 BC
B 1 Ø
Ø 0 Ø
Ø 1 Ø
END
```

Atahan Çaldır

S018194