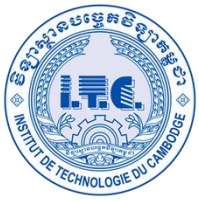


Institute of Technology of Cambodia

Department of Information

Technology and Communication

Technical Report

Group Project

Algorithm and Programing & Automata Theory

Group A4

Lecturer	Subject
BOU CHANNA	**Algorithm and Programming**
VALY DONA	**Automata Theory**

Academic Year 2021-2022

Group Member

N0.	Name	ID	Gender
1	PEOU SOKLEANG	e20190770	M
2	PHENG MALE	e20190813	M
3	LY SIVMENG	e20190567	M
4	HENG CHHAYHONG	e20190263	M
5	HENG VISOTHI	e20190282	M

Contents

I. Introduction

This report is an overview of our project that is about Algorithm and Programing & Automata Theory. The goal is to develop an application to manage and manipulate finite automata by applying what we have learned in Algorithm and Programming class such as linked-list, stacked and queue…and the basic theory on Automata Theory.

II. Function and Feature

There are 6 main functionalities in this program, those are listed below.

1. Design FA

This feature is about insert the data of Finite Automata in string form then convert these strings into data to QueueLL(in data structure) to use for other features and function and also display the table.

EX:

```
1. Design a Finite Automation(FA).
Enter number of state: 3
Start state have be: q0
Final state : q1,q2
Symbol: a,b
Input the transition function all 3 states
q0: q0;q1
q1: q1,q2;q0;q1
q2: q2;q2;q0
```

-Final state and symbol separate by comma ,

+For transition function:

-Use semicolon ; when enter transition for next symbol

-Use comma , when a symbol can cause to have more then one transition

-If no transition put ; before move to the transition by next symbol

After display this table, it will ask to save with an ID or not and also able view all saved FA.

2. Test if FA is deterministic or non-deterministic.

Check if the input FA is deterministic or non-deterministic.

3. Test if a string is accepted by FA.

After enter FA data or Load FA, this feature allows the user to input a string to test if FA accept the string or not

4. Construct an equivalence DFA from an NFA.

When FA is non-deterministic, this function will convert it into an equivalence DFA, also even if FA is DFA it still works but it will remove the non-accessible state and possible to return new form of DFA and represent the same language as old DFA.

5. Minimize a DFA

To activate this function, FA have to be DFA, if it is NFA have to convert it first, then it will do the minimization and return new DFA if DFA can be minimized.

6. Load, Delete and Update FA.

With this function, by input an ID of a saved FA, user can load to reuse, delete and update FA, also able to see all saved FA.

III. Data structure

Array is by far easier to use then Queue but after thinking for a while we decided to use

Queue for our main structure with method FIFO, as we see so many positive points as it is dynamic and we have modified it by calling QueueL can store Queue as it data and moreover we add QueueLL that can store QueueL as its data. So we don't have to worry about size anymore.

Queue:

```
struct Element{
    string data;
    char c;
    int num;
    Element*next;
};

struct Queue{
    int n;
    Element *myrear,*myfront;
};
```

QueueL:

```
struct ElementL{
    Queue* datal;
    ElementL*next;
};


struct QueueL{
    int nl;
    ElementL *myrearl,*myfrontl;
};
```

QueueLL:

```
struct ElementLL{
    QueueL* datall;
    ElementLL*next;
};


struct QueueLL{
    int nll;
    ElementLL *myrearll,*myfrontll;
};
```

These are so similar to Queue structure.

To make it easier to use, so we store FA differently, first we consider FA input is NFA so we use QueueLL, means a QueuL responsible for a state transition by all symbol include epsilon

Then we check if it is DFA then we turn QueueLL into QueueL, means a Queue responsible for a state transition

EX:

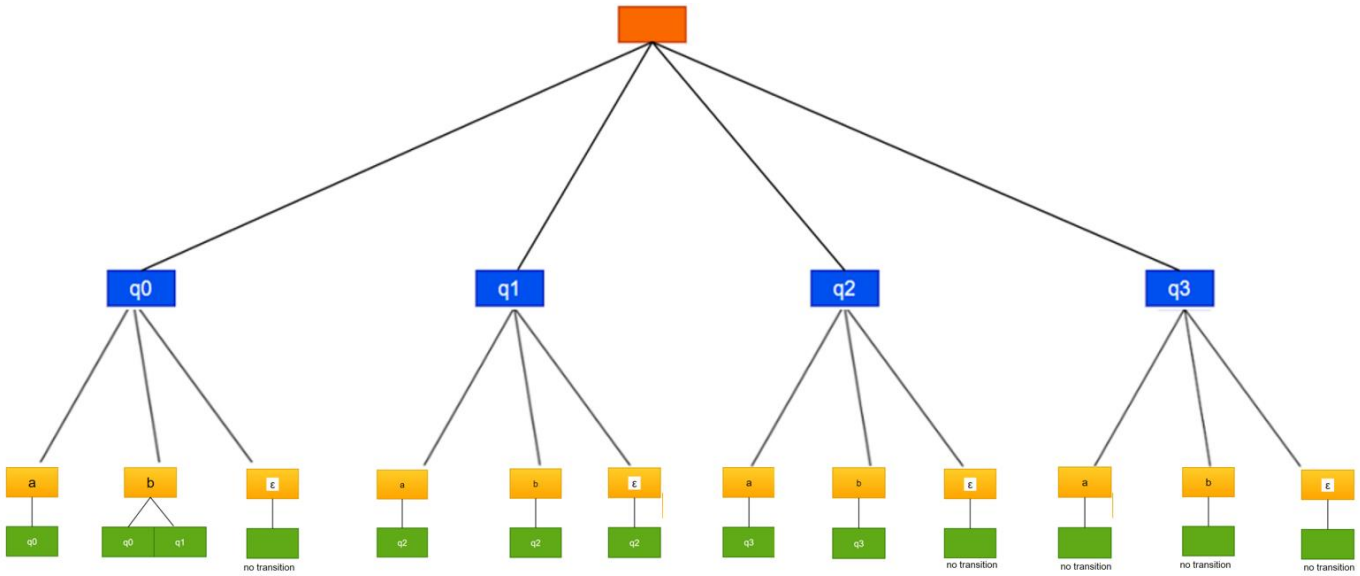
For NFA

QueueLL : Store all state transitions.

QueueL : Store a state transition, its data are Queue, because a symbol can have more transition, so it is set, its first element store the transition by 1st symbol.

Queue : Queue can have zero or more elements base on transition of each symbol.

: transition by any symbol, if no transition it is empty .

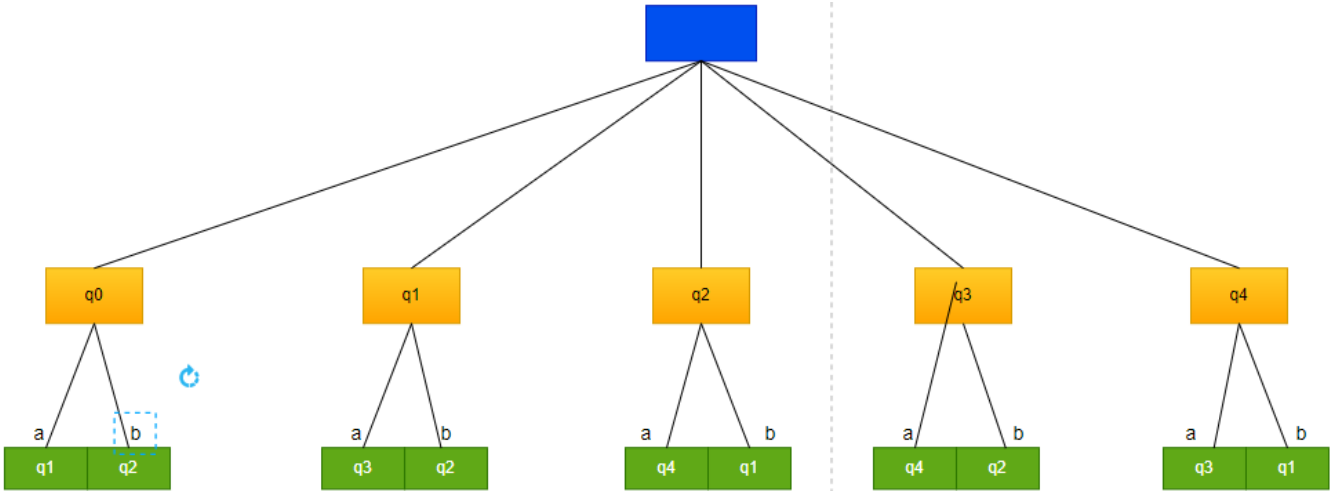


For DFA

QueueL : Store all state transitions .

Queue :Store a state transition, its data are string. Using Queue for a state transition because a symbol can only have one transition so 1st element store the transition by 1st symbol.

: transition by any symbol.



IV. Database Design

In our program, we use txt file to store data of FA as it is the simplest and easiest to use with our current knowledge.

The only solution that we can think is one txt file store one FA data so 100 FAs need 100 txt files.

And we have one file to store all IDs of saved FA and add condition to make sure this file can not be modified from the program.

	a	b	E
->q0	q1	q5	
q1	q6	q2	
*q2	q0	q2	
q3	q2	q6	
q4	q7	q5	
q5	q2	q6	
q6	q6	q4	
q7	q6	q2	

#This table can only be created unless we got all the FA data into QueueLL from Design or Load an ID from a txt file.

V. Implementation

1. Design FA

```
1. Design a Finite Automation(FA).
Enter number of state: 3
Start state have be: q0
Final state : q1,q2
Symbol: a,b
Input the transition function all 3 states
q0: q0;q1
q1: q1,q2;q0;q1
q2: q2;q2;q0
```

After having the data of FA in string form, we start convert these string into new data form using StoreNFA function by putting these into QueueLL. It can be used for other feature and also for display.

FA Table

	a	b	E
->q0	q0	q1	
*q1	q1 q2	q0	q1
*q2	q2	q2	q0

#This table can only be created unless FA data are store in storeNFA(). Just like structure above and for final state and symbol we have other 2 Queue to store them that most the time we name them q2 and q3 in code.

From here there are 4 main functions that we created:

void storeNFA(): store FA data, in state form

void storeDFA(): store FA data when FA is DFA in state form

Then convert the data into(state) into index form, q0=0,q1=q1,….,qn=n.

int stateDFA(): return the index of state when transition by any symbol to any state.

Queue*stateNFA(): return the set of index of state when transition by any symbol to any state.

2. Test if FA is deterministic or non-deterministic

By checking the Queue of each state that store transition of symbol of FA as NFA form,

if its last element (myrearl) is empty(so it has epsilon transition) then it is NFA,

if it's is empty, then check other Queue, if they don't have one transition then FA is NFA otherwise it is DFA.

3. Test if a string is accepted by FA

We use the function from Construct an equivalence DFA from an NFA because it's is complete first. We find that we can use the function from it.

By using function from ConvertNFAToDFA, and recursive which stop when the function when processing the last symbol of string.

- void Process_String(): start from start state, do Eclosure on it then we have a new set

-calling this function again then get the 1st symbol of the string into sstate() function then have new set, keep doing this until reach last symbol, the recursive stop then check if the last result from sstate has at least one final state or not, if it has so The string is accepted or not then The string is not accept.

4. Construct an equivalence DFA from an NFA

Just like the method that we have learned, we start by creating the function to use in this feature

-void EClose0(): input an index state to find its epsilon transition or not, doing recursively until no new epsilon transition is found.

-void EClose(): input a set of state to find their epsilon transition by using EClose0 doin the loop in the set then put the result together without duplicate and also sort them.

-void sstate(): input a set of state to find all their transition by a specific symbol, then we will have new set, next we take this new set to do EClosure with the ECose() function.

To the main point of converting:

-void ConvertNFAtoDFA(): use recursive where base case to stop condition is when no new set of state is found or it is NULL in QueueL*N just like in the lesson.

-start from start state, use EClose to do Eclosure on start state then we have new set of state, in lesson we call it q'0, and remember this set when new set same as this set, let store it in N (N is QueueL).

-Call this convert function again by retrieve the set that we store in N start from 1st element to do loop base on number of symbol and call sstate function, so its result is a set, check if this set is new, add it to N also add it to new QueueL, let call it Q0 to store new DFA, if not then check what index is it in N, then add to Q0..

-keep doing this until no new set is found so recursive stop and we have a new DFA and N that have all sets of state that we get from using sstate. Then with last function we just get the final state from N and reduce some argument which are not necessary to have.

#This new DFA data are index form, so we just convert into state form when we display table

5. Minimize a DFA

By using the method that we have learned, we also create function to use this minimization to.

First store FA into DFA form, if it's NFA, convert first.

-void Non_accessible_state(): get non-accessible state and accessible state where start state can not be non-accessible state by check it each state if is not appear in transition function of other state then it is non-accessible state.

-void Distinguishable_a_pair(): check a pair is distinguishable or not, a pair have number so check this if one of them is final state then it is distinguishable.

-void Distinguishable_all_pair(): start by create triangle table then exclude non-accessible pair after that get all pair that Distinguishable and pair that are not distinguishable to examine them later by using Distinguishable_a_pair() function.

- Queue* unmarked_pair() : juts to get new pair when transition by a symbol

- void examine_all_unmarked_pair(): do recursively until no new mark pair is found by using unmaked_pair(), then check it result if it is in marked(distinguishable) pair or not , if it is, add to marked pair.

- void getEquivalence_pair(): get the remaining unmarked pair or equivalence pair

- void groupEquivalence(): group all the equivalence pair

- void GroupNew_State(): use the equivalence pair to group new set for new DFA

-void DFA_After_Minimize(): get new DFA here but no final state yet will combine all function above into one,

- void Minimization(): combine all function above, then we have new DFA and new Final state.

#This new DFA data are index form, so we just convert into state form when we display table

6. Load, Delete and Update FA

The ways we store FA data is almost same as how we let the user input the FA data

-1st line is for symbol

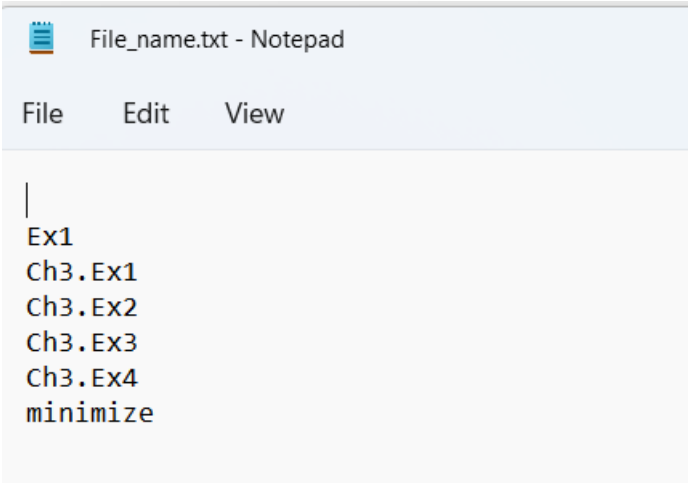
-2nd line is for final state

-from 3rd line are the transition by 1st state to last state

Then use them to store back in using sotreNFA, Queue then we have the data that can be use for other feature.

To Load, delete, update is just access to ID(file name) but to see all FA data, we store all FA'ID(file name) into other txt file that we name it "File_name" and put the condition to make sure this file can never be modified from our program.

Then we when delete by not just delete the txt file, we have to delete the ID that store in file name by overwrite the old data and exclude the deleted ID.



VI. Result

The program able to do all the 6 functionalities

-User can input FA data and save

-User can check if FA is NFA or DFA

-User can input string to test FA

-User can convert FA when it is NFA to DFA

-User can minimize FA when it is DFA

-User can load, delete and update saved FA.

VII. Conclusion and Perspective

As we only work hard in the last few weeks, we still expect the missing part or bug as we messed up, we didn't rush enough to start working hard on project and only have reach Test string FA during progress follow up.

Now it's is finished. We become much clear about Automata theory and Queue because we use Queue. We hope not to mess up like this again in future.