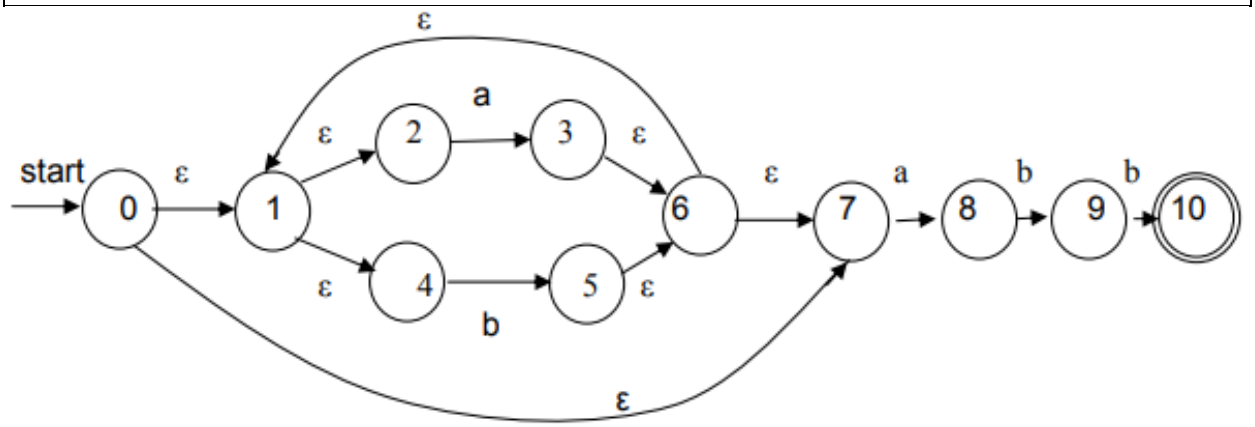
Suez Canal University

Faculty of **Faculty of Computers**

Techniques used in constructing DFA from NFA



Student name	Amir Haytham Muhammad Salama
Faculty	Faculty of Computers and Informatics
Level	Third Level
Department	Computer Science
National ID	29906261900473
Student code	1714103022
Program	Computer Science Program
Course	Compiler Design

1. Introduction

In my research, I have asked about the techniques that are used in constructing DFA from NFA. But at first let's introduce this. In the theoretical foundations of computer science, automation theory is the study of abstract machines and the problems they can solve. These abstract machines are called automatic machines. A discrete automation is a mathematical model for a finite state machine (FSM). The FSM is a machine that takes the symbols as inputs and transitions, from one state to another according to a transition function (which can be represented as a transition table). This transition function tells the automation which state to go to next given the current state and current symbol. Turing Machine is one of the vital portions of Automata Theory; it's the father of all computers. That is Automata Theory a set of mathematical computations and formulas explaning the automation or process of that machine. Automata Theory does not deal with actual automatons, such as robots, but deals with a virtual entity on a machine. [3,4,5]

The automation reads each character as it moves through states. Combined characters make up a list when the automaton ends at an agreed condition. Basically two models are discussing. Of which, One model called finite automation is used for text processing, compilers, and hardware design. Another model, called context-free grammar, is used for programming languages and artificial intelligence. Automata theory is an excellent place to continue researching the theory of computing. Computability and complexity theories require a specific description of a machine. Automata Theory enables practice with formal definitions of computation as it introduces concepts relevant to other non-theoretical areas of computer science. [1:6]

2. Research items

- The main idea of Converting NFA to DFA
- Converting From Regular Expression to NFA to DFA
- Regular expression
- Converting the NFA into a DFA

- NFA vs. DFA (Time-Space Tradeoffs)

3. Research

- # The main idea of Converting NFA to DFA: [1:17]

Given: A non-deterministic finite state machine (NFA).

Goal: Convert to an equivalent Finite State Deterministic Machine (DFA)

Why? Faster recognizer!

Approach: Consider simulating a NFA. Work with sets of states**.**

IDEA: Every state in the DFA corresponds to a set of NFA states.

Worst-case: There might be an exponential number $O(2^n)$ of a set of states. The DFA may have several more states exponentially than the NFA, but this is unusual.

- **Algorithm: Convert NFA to DFA:**

We'll use:

Move $_{NFA(S,a)}$	the transition function from NFA
ε -Closure(s)	where s is a single NFA state
ε -Closure(S)	where S is a group of states from NFA

We'll construct:

S_{DFA} the set of states in the DFA. Initially, we'll set SDFA to {}

Step 1: Initially Q' = ϕ

Step 2: Add q0 of NFA to Q', and the we have to find the transitions from start state.

Step 3: In Q', find the possible group of states for each input symbol. If this group of states is not in Q', then add it to Q'.

Step 4: So in DFA, the final state will be all the states that contain F(final states of NFA)

• Converting Regular Expression to NFA to DFA:

It has been shown (Kleene's Theorem) that RE and FA are similar language description methods. Based on this theoretical result, functional algorithms have been established that allow us to actually create FA's from RE's and simulate the FA with a computer program using Transition Tables.Following this process, the NFA is first built from a regular expression, then the NFA is rebuilt into a DFA, and finally the Transformation Table is built. The Thompson Construction Algorithm is one of the algorithms that can be used to build a Nondeterministic Finite Automaton (NFA) from RE, and the Subset Construction Algorithm can be used to transform the NFA to a Deterministic Finite Automaton (DFA). The last move is to build a transition table. We need a finite state machine that is a deterministic finite automation (DFA) so that each state has a unique edge for the input alphabet element. And there's no confusion for the generation of code. But a non-deterministic finite automation (NFA) with more than one edge for an input alphabet element is easier to construct using a general algorithm-Thompson 's construction. After a normal process, we convert the NFA to the DFA for coding. [2:6,9,10]

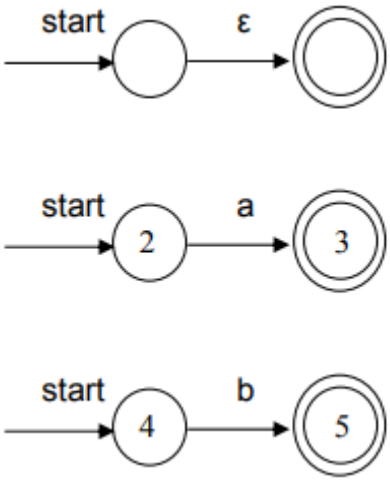
• Regular Expression:

Consider a regular expression r = (a|b)*abb, that matches {abb, aabb, babb, aaabb, bbabb, ababb, aababb,……} To buid the NFA from this, use Thompson’s

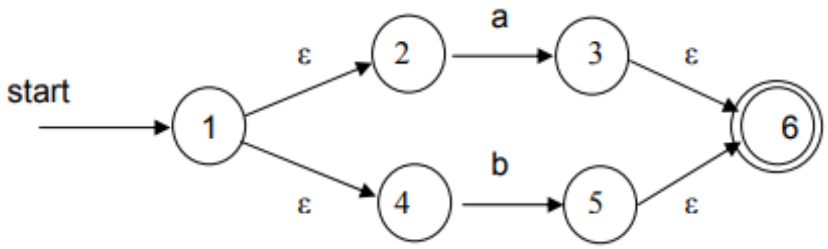
construction. This method constructs a regular expression from its components using the ε-transitions. For the NFA subcomponent, the ε transitions act as a "glue or mortar." The ε-transition adds nothing as the concatenation with the empty string leaves a regular expression unchanged (the operation of identity is concatenation with ε). Step 1, parse this regular expression into its subexpressions involving alphabet symbols a and b and ε: ε, a, b, a|b, ()*, ab, abb These describe a. the regular expression for single characters π, a, b.. alternation between a and b representing the union of the sets: L(a) U L(b) c. Kleene star ()* d. Concatenation of a and b: ab, and even ab Subexpressions of this kind have their own nondeterministic Finite Automata from which the overall NFA is constructed. Each NFA component has its own start and end state acceptance. A nondeterministic Finite Automata (NFA) has a transition diagram of probably more than one edge for a symbol (character of the alphabet) that has a starting state and an accepting state. The NFA certainly provides an accepting state for the symbol. [1,2,3,5,7]

- **Take these NFA's process:**

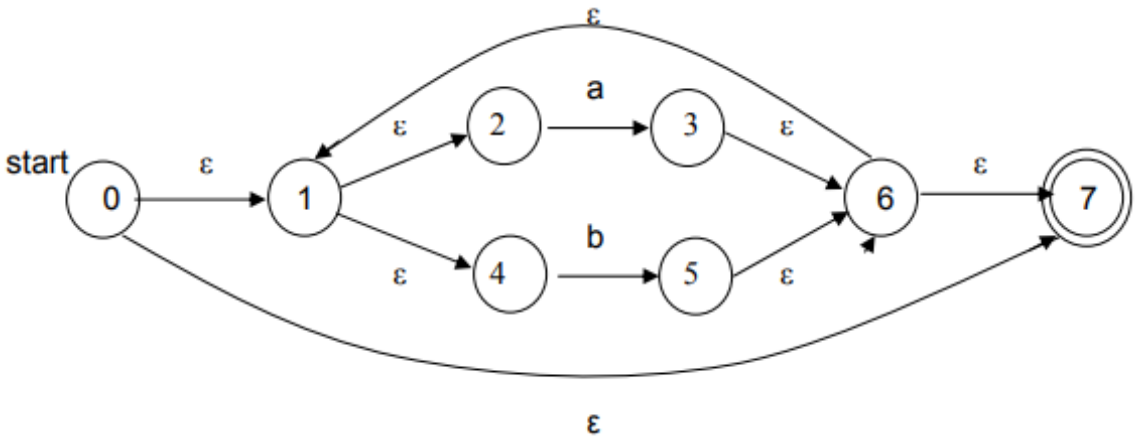
  a. The NFA's for the single character regular expressions ε, a, b



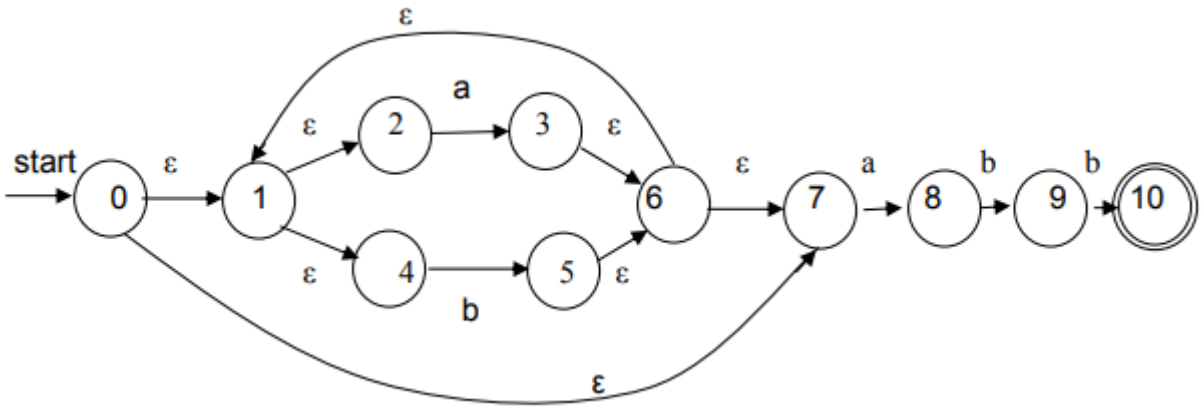
b. The NFA for the a and b union: a|b is constructed from the individual NFA's using the ε NFA as "glue". Drop the individual accepting states and replace with the general accepting state



c. Kleene star on (a|b)*. The NFA accepts ε providing with (a|b)*



d. Concatenate to abb



This is the complete NFA which is describing t he regular expression (a|b)*abb. The problem is that it is not ideal as the basis for a DFA transition table, as multiple edges leave several states (0, 1, 6).

- # Converting the NFA into a DFA: [7:15]

A Deterministic Finite Automaton (DFA) has a maximum edge for a given symbol from each state, and is an appropriate basis for a transition table. We need to eliminate the ε-transitions by building a subset.

- **Definitions:**

Consider a single state s. Consider a set of states T,

Operation	Description
ε-closure(s)	Set of NFA states reachable from NFA state **s** on ε-transitions alone
ε-closure(T)	Set of NFA states reachable from set of states **T** on ε-transitions alone
move(T,a)	Set of states to which there is a transition on input symbol **a** from some NFA state in **T**

We have the set of N states as its entry. In a DFA we generate a set of D states as output. An NFA with n states can theoretically generate a DFA with 2^n states.

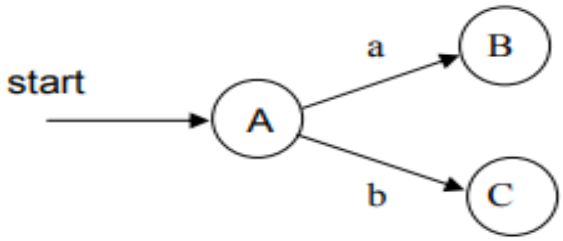
- **Start the Conversion**

1. Start with starting state 0 and calculate ε-closure(0).

   a. {0,1,2,4,7} is the set of states that can be reached by ε-transitions which includes 0 itself.

   A = {0,1,2,4,7}

2. We must now find the states that A connects to. There are two symbols in the language (a, b) so we expect only two edges in the DFA: from A to a, and from A to b. Call these states B and C:



- **We find B and C in the following way:**

Find the state B with an edge of a from A:

a. Start with A{0,1,2,4,7}. Find which states in A have states reachable by the transitions. The current set is called move(A,a) The set is {3,8}: move(A,a) = {3,8}

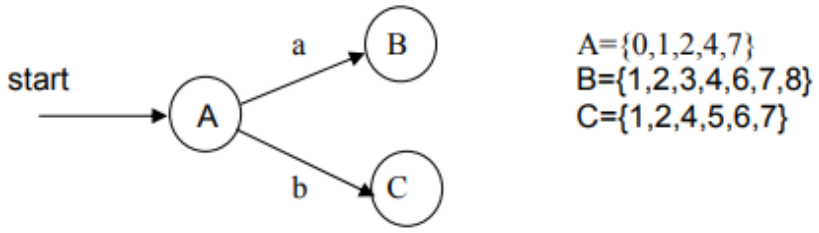
b. Now do an ε-closure on move(A,a). By Finding all the states in move(A,a) which are reachable with ε-transitions. We have 3 and 8 to consider. By Starting with 3, we can get to 3 and 6 and from 6 to 1 and 7, and from 1 to 2 and 4. By Starting with 8, we can get to 8 only. So, the completed set is {1,2,3,4,6,7,8}. Therefore ε-closure(move(A,a)) = B = {1,2,3,4,6,7,8} This can define a new state B that has an edge on a from A

Find the state C which has an edge on b from A:

c. By starting with A{0,1,2,4,7}. We have to find which states in A have states reachable by b transitions. This current set is called move(A,b) The set is {5}: move(A,b) = {5}

d. Now, do an ε-closure on move(A,b). By Finding all the states in move(A,b) which are reachable with ε-transitions. We have only

state 5 to consider. From 5 we get to 1, 2, 4, 5, 6, 7. So the complete set is {1,2,4,5,6,7}. So ε-closure(move(A,a)) = C = {1,2,4,5,6,7} This can define a new state C that has an edge on b from A

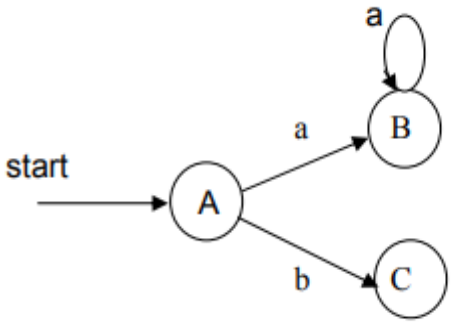


Now we have B and C, and we can move on to find the states that have a and b transitions from B and C.

Find the state which has an edge on a from B

e. By starting with B{1,2,3,4,6,7,8}. Find which states in B have states reachable by the transitions. This current set is called move(B,a) The set is {3,8}: move(B,a) = {3,8}

f. Now, do an ε-closure on move(B,a). So we have to find all the states in move(B,a) which are reachable with ε-transitions. We have 3 and 8 to consider. By starting with 3 we can get to 3 and 6 and from 6 to 1 and 7, and from 1 to 2 and 4, and by starting with 8 we can get to 8 only. Therefore the complete set is {1,2,3,4,6,7,8}. Hence ε-closure(move(A,a)) = {1,2,3,4,6,7,8} **which is the same as the state B itself.** To put it another way, we have a repeating edge to B:

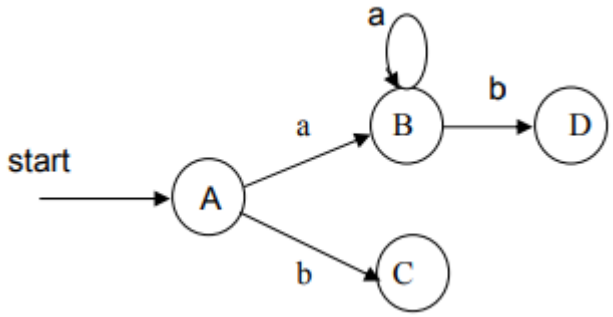


A={0,1,2,4,7}
B={1,2,3,4,6,7,8}
C={1,2,4,5,6,7}

Find the state D with an edge of B on b:

g. By starting with B{1,2,3,4,6,7,8}. So we have to find which states in B have states reachable by b transitions. This current set is called move(B,b) The set is {5,9}: move(B,b) = {5,9}

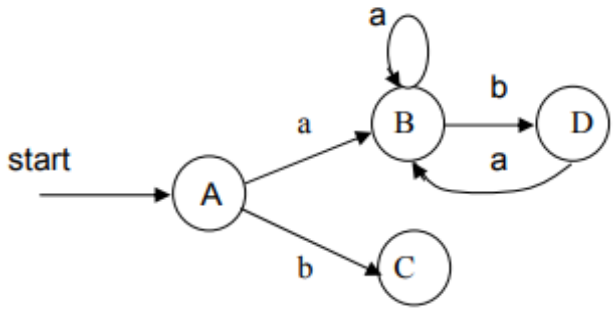
h. Now, do an ε-closure on move(B,b). So we have to find all the states in move(B,b) which are reachable with ε-transitions. From 5, we get to 1, 2, 4, 5, 6, 7. From 9, we get to 9 itself. Hence, the complete set is {1,2,4,5,6,7,9}}. So ε-closure(move(B,a)) = D = {1,2,4,5,6,7,9} This can define the new state D that has an edge on b from B A={0,1,2,4,7}, B={1,2,3,4,6,7,8}, C={1,2,4,5,6,7}, D{1,2,4,5,6,7,9}



Find the state with an edge on a from D:

i. By starting with D{1,2,4,5,6,7,9}. We have to find which states in D have states reachable by the transitions. This current set is called move(D,a) The set is {3,8}: move(D,a) = {3,8}

j. Now, do an ε-closure on move(D,a). We have to find all the states in move(B,a) which are reachable with ε-transitions. We have 3 and 8 to consider. By starting with 3, we get to 3 and 6 and from 6 to 1 and 7, and from 1 to 2 and 4, and by starting with 8 we can get to 8 only. Hence the complete set is {1,2,3,4,6,7,8}. Therefore ε-closure(move(D,a)) = {1,2,3,4,6,7,8} = B This is a return edge to B: A={0,1,2,4,7}, B={1,2,3,4,6,7,8}, C={1,2,4,5,6,7}, D{1,2,4,5,6,7,9}



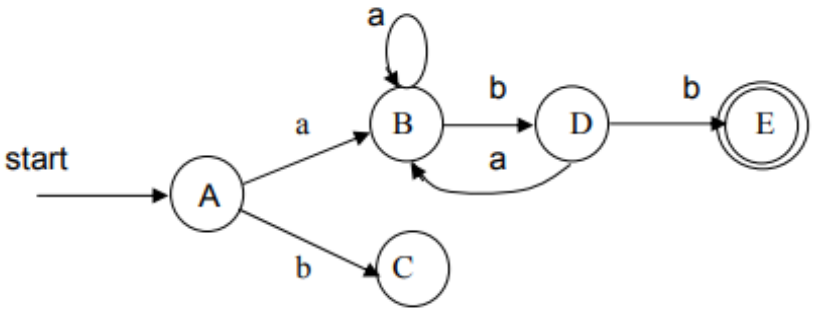
Find the state E with an edge of D on b

k. By starting with D{1,2,4,5,6,7,9}. We have to find which states in D have states reachable by b transitions. This current set is called move(B,b) The set is {5,10}: move(D,b) = {5,10}

l. Now, do an ε-closure on move(D,b). We have to find all the states in move(D,b) which are reachable with ε-transitions. From 5, we get to 1, 2, 4, 5, 6, 7. From 10, we get to 10 itself. Hence the complete current set is {1,2,4,5,6,7,10}}. Therefore ε-closure(move(D,b) = E = {1,2,4,5,6,7,10} This can define the new state E that has an edge on b from D.

Since, it has an accepting state, it is an accepting state too.

A={0,1,2,4,7}, B={1,2,3,4,6,7,8}, C={1,2,4,5,6,7},
D={1,2,4,5,6,7,9}, E={1,2,4,5,6,7,10}

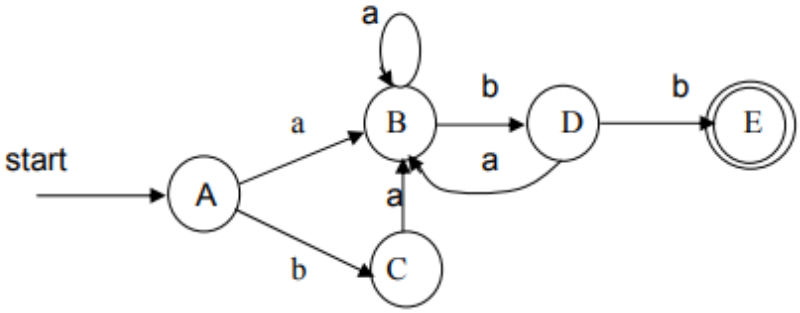


We should now examine state C

Find the state that has edge on a from C:

m. By starting with C{1,2,4,5,6,7}. We have to find which states in C have states reachable by the transitions. This current set is called move(C,a) The set is {3,8}: move(C,a) = {3,8} we have seen this before. It's the state B

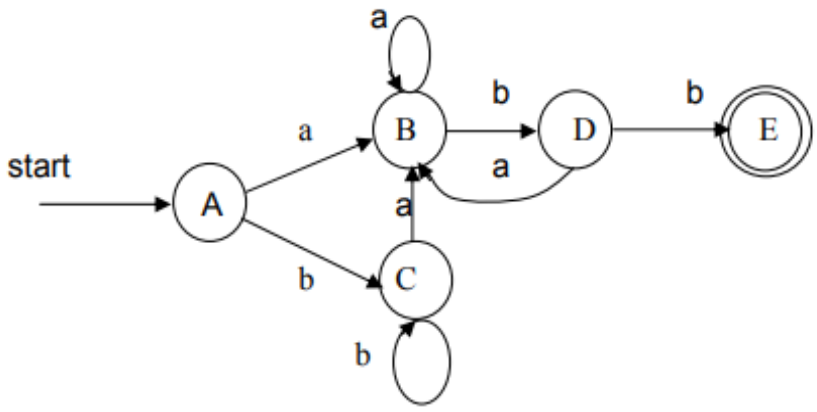
A={0,1,2,4,7}, B={1,2,3,4,6,7,8}, C={1,2,4,5,6,7},
D={1,2,4,5,6,7,9}, E={1,2,4,5,6,7,10}



Find the state which has an edge on b from C:

n. By starting with C{1,2,4,5,6,7}. We have to find which states in C have states reachable by b transitions. This current set is called move(C,b) The set is {5}: move(C,b) = {5}

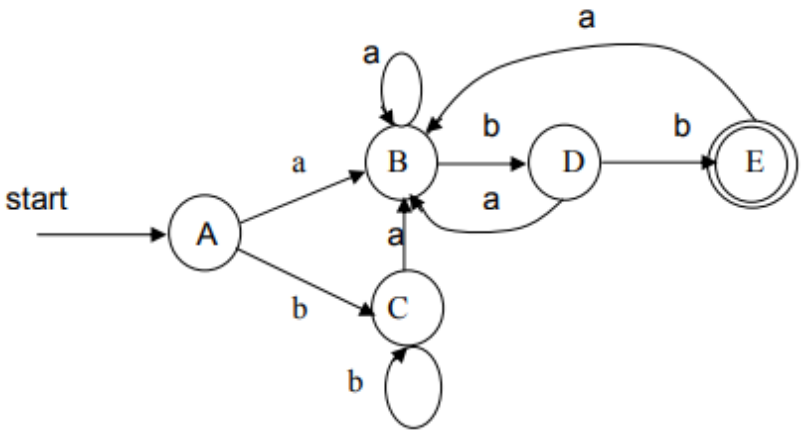
o. Now, do an ε-closure on move(C,b). We have to find all the states in move(C,b) which are reachable with ε-transitions. From 5, we can get to 1, 2, 4, 5, 6, 7. which is C. So ε-closure(move(C,b)) = C. This can define a loop on C



Finally, we have to look at E. Although this is an accepting state, the regular expression provides us to repeat adding in more a's and b's as long as we return to the accepting E state finally. So,

Find the state with an edge on a from E:

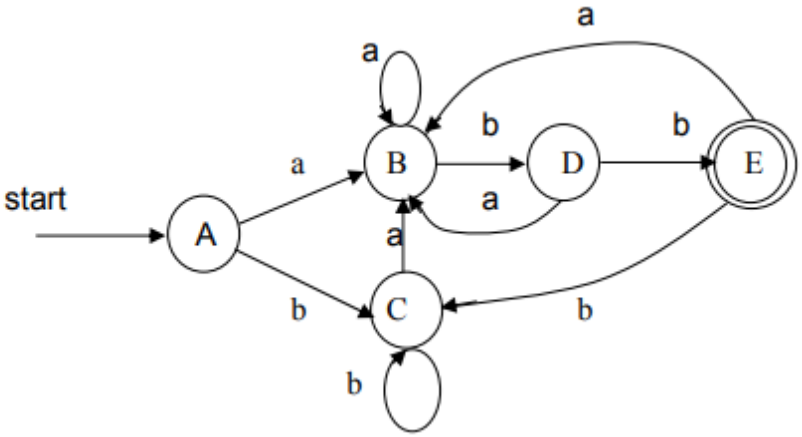
p. By starting with E{1,2,4,5,6,7,10}. We have to find which states in E have states reachable by a transitions. This current set is called move(E,a) The set is {3,8}: move(E,a) = {3,8} We have seen this before, it's B. So,



Find the state with an edge of E on b:

q. By starting with E{1,2,4,5,6,7,10}. We have to find which states in E have states reachable by b transitions. This current set is called move(E,b) The set is {5}: move(A,b) = {5}

We've seen this before. It's C. Finally,



Here we are! For a given input character there is only one edge of each state. It's a DFA. Ignore the fact that each of these states is also a group of NFA states. In the DFA, we should find them as single states. In fact, it requires other as an edge beyond E leading to the ultimate accepting state too. Also, the DFA is not optimized (there

can be less states). However, we can make the transition table. Here it is:

State	Input **a**	Input **b**
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

- # NFA vs. DFA (Time-Space Tradeoffs) [7:15]

The following table summarizes the worst-case to determine if an input string x belongs to the language denoted by a regular expression r using recognizers constructed from NFA and DFA.

Automaton	Space	Time
NFA	$O(\|r\|)$	$O(\|r\| \times \|x\|)$
DFA	$O(2^{\|r\|})$	$O(\|x\|)$

| r | is the length of r, and | x | is the length of x.

Example:

For the following regular expression:
(a | b)*a(a | b)…(a | b), where there are n - 1 (a | b)'s at the end
There is no DFA with fewer than number 2^n states.

4. Conclusion

So, I have introduced my problem in theoretical foundation, and I have given at first pseudocode to start with it As converting NFA to DFA. Also, I have attached some graphs and tables to make my explanation more readable and understandable. At the end, I have made a comparison between NFA and DFA, I have mentioned the tradeoffs, complexity time order, and space order.

5. References

1. Murugesan, N., & Sundaram, O. S. A General Approach to DFA Construction.

2. van Zijl, L., Harper, J. P., & Olivier, F. (2000, July). The MERLin environment applied to⋆-NFAs. In *International Conference on Implementation and Application of Automata* (pp. 318-326). Springer, Berlin, Heidelberg.

3. Cooper, K., & Torczon, L. (2011). *Engineering a compiler*. Elsevier.

4. Domaratzki, M., & Salomaa, K. (2006, August). Lower bounds for the transition complexity of NFAs. In *International Symposium on Mathematical Foundations of Computer Science* (pp. 315-326). Springer, Berlin, Heidelberg.

5. Wulf, W. A. (1981). Compilers and computer architecture. *Computer*, (7), 41-47.

6. Sidhu, R., & Prasanna, V. K. (2001, March). Fast regular expression matching using FPGAs. In *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)* (pp. 227-238). IEEE.

7. Wikipedia: https://en.wikipedia.org/wiki/Powerset_construction
8. GeeksForGeeks: https://www.geeksforgeeks.org/conversion-from-nfa-to-dfa/
9. Javat tutorial point: https://www.javatpoint.com/automata-conversion-from-nfa-to-dfa

10. YouTube: https://www.youtube.com/watch?v=pnyXgIXpKnc
11. YouTube: https://www.youtube.com/watch?v=--CSVsFIDng
12. YouTube: https://www.youtube.com/watch?v=pnyXgIXpKnc
13. YouTube: https://www.youtube.com/watch?v=syjJutOdLnI
14. YouTube: https://www.youtube.com/watch?v=taClnxU-nao
15. Tutorialspoint: https://www.tutorialspoint.com/automata_theory/ndfa_to_dfa_conversion.htm