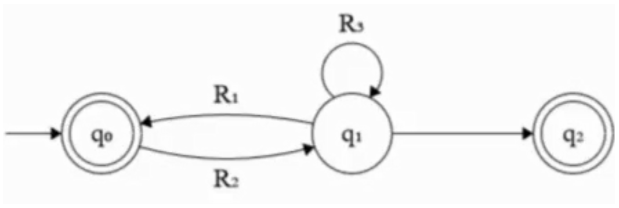
CSC301

Lecture 8 summary

1910456

An interesting problem from a student

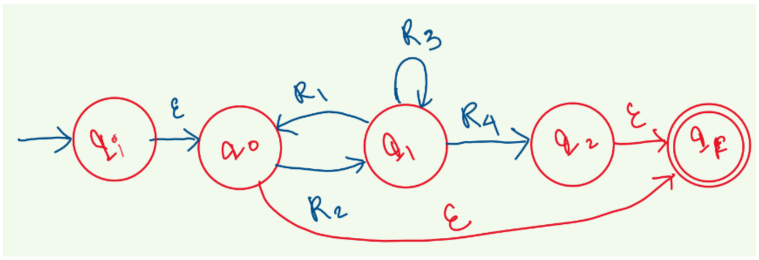


[Fig 1: First Problem]

From the knowledge of last lecture, we can know that initial cannot have arrows coming into it and final cant have arrows moving out from it. And that there must be one initial and final state.

So, we add our own initial and final states. We add the ε-transitions. Make the previous final and initial states normal.

We get:

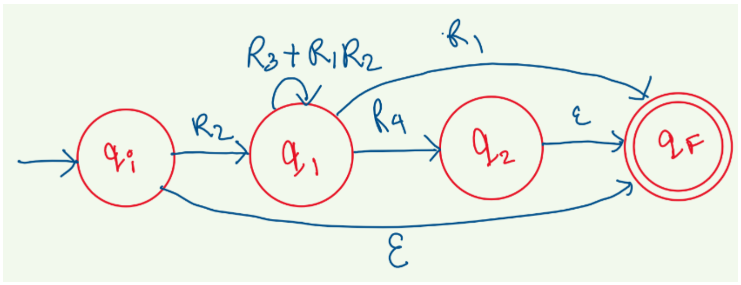


[Fig 2: Added initial and final states]

Then we try to remove q_0, we see that q_1 has a self-loop R_3, but also, we can make a self-loop R_1R_2 and hence remove the R_1 arrow that went from q_1 to q_0.

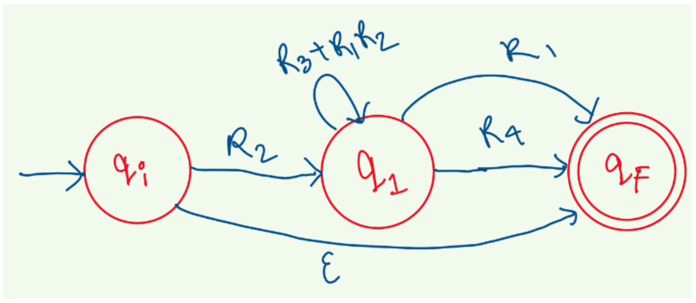
There is also the ε – transition to the final state that we had added. We also see that $(R_2R_1)^*$ from q_0 was an accepted string, therefore we add an R_1 arrow from q_1 that directly goes to q_F.

We get:



[Fig 3: removed q_0]

Since being on q_2 is the same as being in the q_F, we remove q_2.



[Fig 3: removed q_2]

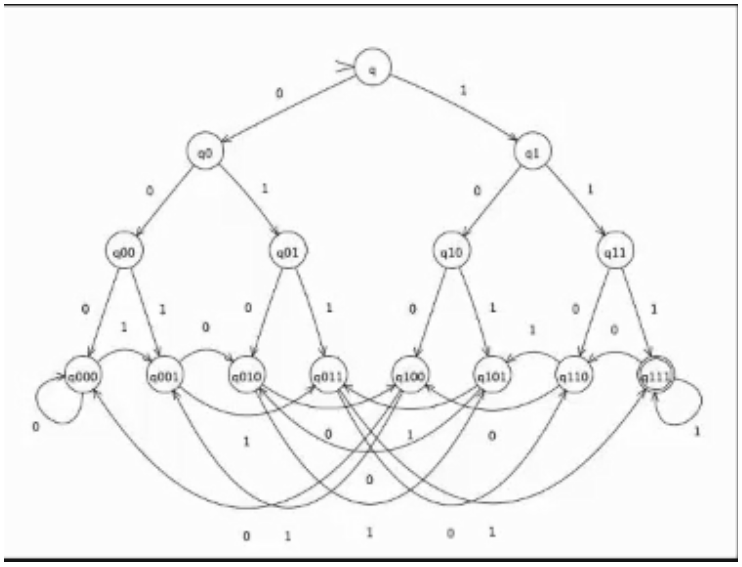
Now all that is left is to remove q_1.

Therefore, the final Regex is:

$\varepsilon + R_2(R_3 + R_1R_2)^*(R_1 + R_4)$

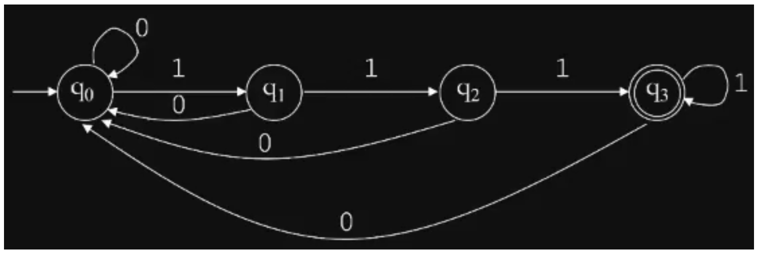
The process with which we make DFAs, (NFA to DFA or Regex to NFA or intuitively), we cannot actually say for sure if the DFA has the minimum number of states possible.

For the gigantic DFA:



[Fig 4: Giant DFA from Algorithmic approach]

We saw that we could simplify it using the intuitive approach:



[Fig 5: Simplified DFA from intuitive approach]

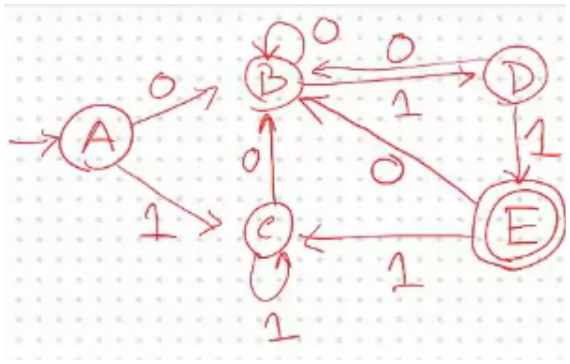
Here, both are “Correct”. Both have the same execution time.

The 1st one is easier to develop than the 2nd.

The second one has a lesser Model Complexity and requires less storage.

But we have to ask if the 2nd DFA really has the minimal number of states possible.

DFA Minimization



[Fig 6: Example 1 DFA]

Firstly, we have to assume that a DFA can at least have two sets.

We call it 0-equivalence, where we group all the non-final states and final states separately.

For Example 1,

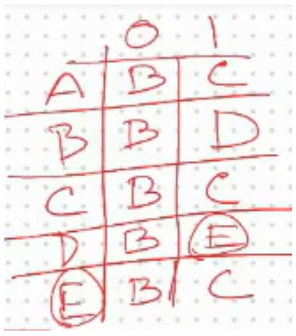
0-equivalence: {A,B,C,D},{E}.

Then we start to refine and see if we can divide the sets further by identifying unique states.

The way we do it is we check if all the states in a set behave the same way when they are fed an input. By that we mean, they all go to the same set of states for each input. (Note: e.g for input 0, they can all move to one set of states, for input 1, they can all move to another set of states. But the important part is that for a given input they all go to the same set of states).

∴ we start by listing the transitions of each state in a set and verify their uniqueness.

It is helpful to use a transition table:



	0	1
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

[Fig 7: Transition table of Example 1]

Then we list the transitions,

{A,0}→{A, B, C, D}, {A,1}→{A, B, C, D}

{B,0}→{A, B, C, D}, {B,1}→{A, B, C, D}

{C,0}→{A, B, C, D}, {C,1}→{A, B, C, D}

{D,0}→{A, B, C, D}, {D,1}→{E}

Here we see that D is behaving differently for input 1.

∴ we take D out of the set:

1-equivalence: {A, B, C}, {D}, {E}

Since there was a change in our states from 0-eq to 1-eq, we have to check again:

{A,0}→{A, B, C}, {A,1}→{A, B, C}

{B,0}→{A, B, C}, {B,1}→{D}

{C,0}→{A, B, C}, {C,1}→{A, B, C}

B is unique here. So,

2-equivalence: {A, C}, {B}, {D}, {E}

We repeat because there was a change,

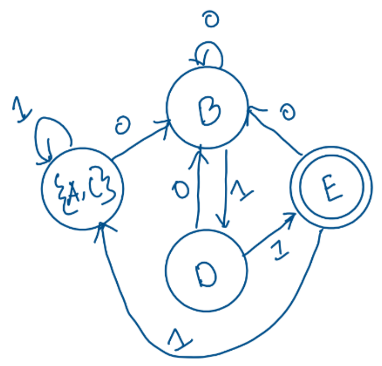
{A,0}→{B}, {A,1}→{A, C}

{C,0}→{B}, {C,1}→{A, C}

Their transitions are similar, therefore no change.

3-equivalence: {A, C}, {B}, {D}, {E}

So now we see that we have taken out all the unique states and are left with four sets of states. Now we can create our minimal state DFA.



[Fig 8: Simplified DFA after minimization]