Finite Automata to Regular Expression

GNFA Method

Slides by Saumya Gupta
CSE303, Stony Brook University
(2021-10-04)

Overview + What is GNFA?

- We will show that any finite automata (DFA/NFA) can be converted into a regular expression.
- We will create a GNFA (Generalized Non-Deterministic Finite Automata) from the given DFA/NFA diagram.
- In DFA/NFA, we are only allowed to write ε or members of the alphabet on the transition arrows. In a GNFA, we are allowed to write regular expressions on the transition arrows.
- The basic idea of converting the DFA/NFA to GNFA is to keep removing states from the given DFA/NFA diagram until we are left with only one START and one FINAL state in the GNFA.
- On removing states, we will write regular expressions on the transition arrows. The final GNFA diagram will have only one transition arrow from one START state to one FINAL state. The regular expression on this transition arrow will give us the answer to the desired regular expression.

GNFA Guidelines

For convenience, we require our GNFA at all times to be of the following form:

1. There is only one START state. This state has no incoming arrows.
    - How to achieve this? Add a new START state with an ε arrow to the old START state.

2. There is only one FINAL state. This state has no outgoing arrows.
    - How to achieve this? Add a new FINAL state with ε arrows from all the old FINAL states.

3. The START and FINAL state are not the same.
    - How to achieve this? The above two steps will ensure that START and FINAL state are not the same.

Top-level outline of conversion

We will convert the DFA/NFA to a regular expression as follows:

1. Convert DFA to GNFA by adding new START and FINAL states as mentioned in the previous slide (slide 3).
2. Remove all states one-by-one, until we have only the START and FINAL state.
3. Output regex is the label on the single transition left in the GNFA.

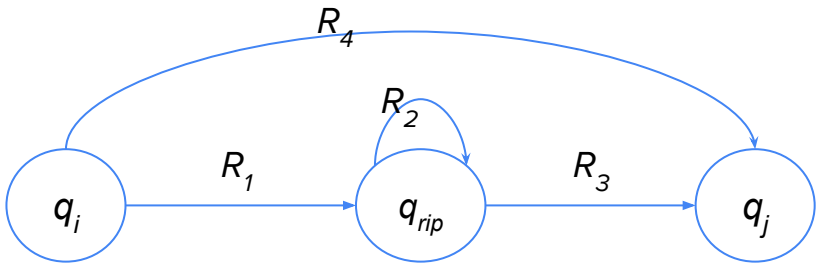
Ripping process : How to remove a state?

At every step, we need to construct an equivalent GNFA with one fewer state, till we only have one START and one FINAL state remaining.

We do so by selecting any state (≠ START, FINAL), ripping it out of the machine, and repairing the remainder so that the language is still recognized. The order in which the states are ‘ripped out’ does not matter because ultimately we need to have ripped out all the states (≠ START, FINAL). Let the ripped out state be q_{rip}.

After removing q_{rip}, we repair the machine by altering the regex on each of the remaining arrows.

Ripping process : How to remove state q_{rip}



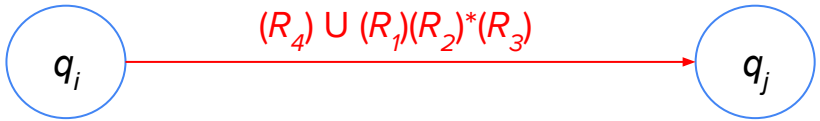
old machine

In the old machine, if

1. q_i goes into q_{rip} with an arrow labeled R_1
2. q_{rip} goes to itself with an arrow labeled R_2
3. q_{rip} goes to q_j with an arrow labeled R_3
4. q_i goes to q_j with an arrow labeled R_4

Then **in the new machine**, the arrow from q_i to q_j gets the label:

$(R_4) \cup (R_1)(R_2)^*(R_3)$

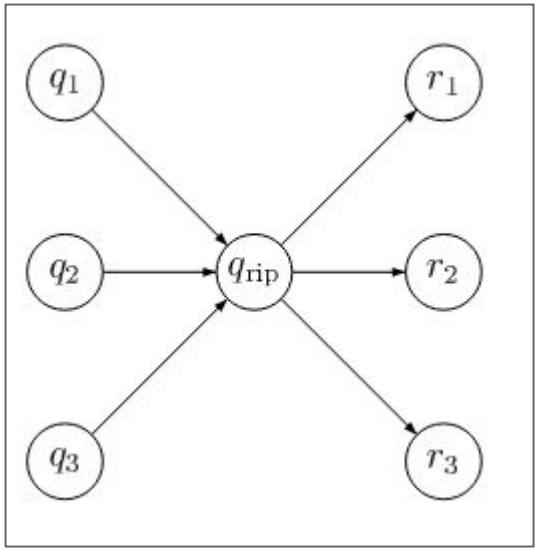


new machine

We have to make this change for each arrow coming from any incoming state q_i to any outgoing state q_j including the case where $q_i = q_j$. The resulting machine is equivalent to the old machine, with one fewer state.

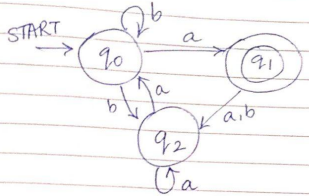
Steps

1. Convert DFA/NFA to GNFA by adding new START and FINAL states as mentioned in slide 3.
2. For every state (≠START,FINAL) we need to remove it from the machine. Let one such state be q_{rip}.
   - For each pair of states q_i and r_i as shown in the figure, we need to convert the transition through q_{rip} into a direct transition from q_i to r_i using the method shown in slide 6.
3. Step 2 is done till we are left with only 2 states, one START and one FINAL state.

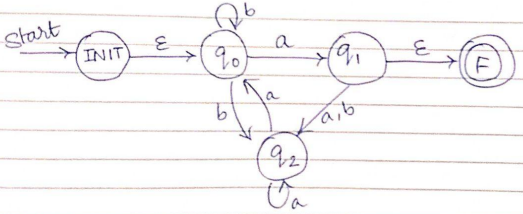


Example 1

(1) Given DFA:



(2) ~~Add~~ ADD NEW START STATE (INIT) AND FINAL STATE (F)



(3) RIP q_2:

MAKE A LIST OF STATES WITH ARROWS GOING TO q_2, AND STATES WHICH q_2 HAS OUTGOING ARROWS TO.

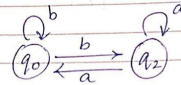
THUS, "IN" STATES : q_0, q_1
"OUT" STATES : q_0

THUS, WE NEED TO FIX REGEX ON FOLLOWING ARROWS:

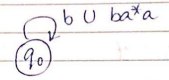
$q_0 \rightarrow q_0$
$q_1 \rightarrow q_0$

THAT IS, WE FIX ARROWS BETWEEN EVERY PAIR OF STATES (q_i, q_j) SUCH THAT $q_i \in$ "IN" LIST AND $q_j \in$ "OUT" LIST

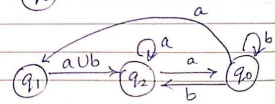
FIXING $q_0 \rightarrow q_0$: old :



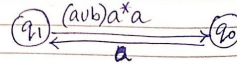
USING SLIDE 6 : new



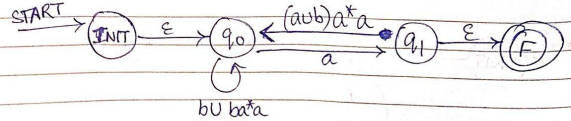
FIXING $q_1 \rightarrow q_0$: old :



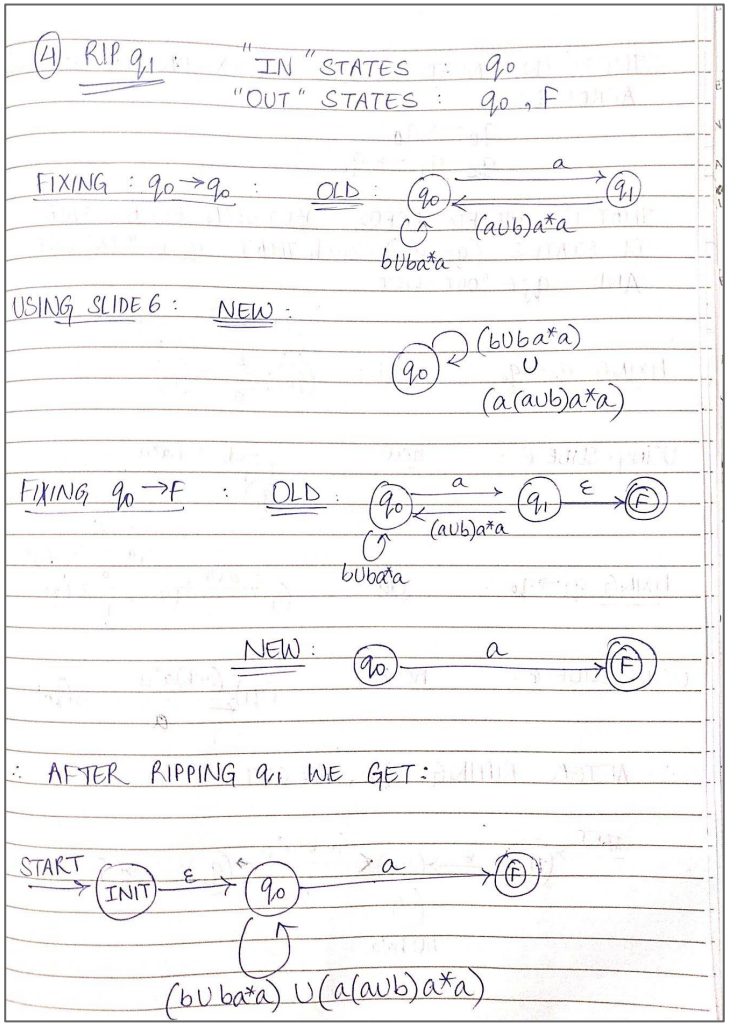
USING SLIDE 6 : new :

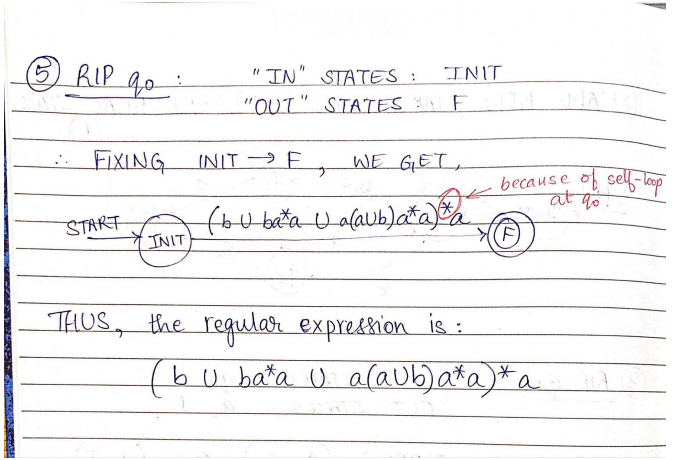


∴ AFTER RIPPING q_2, WE GET:



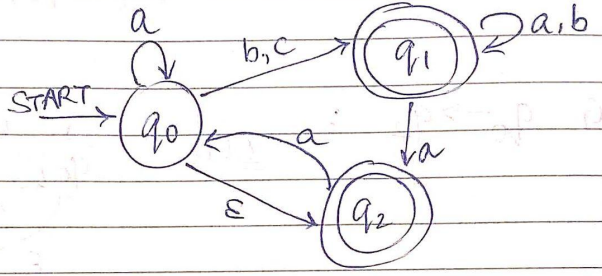
Example 1 (contd)



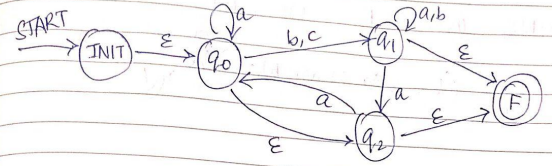


Example 2

① GIVEN NFA :



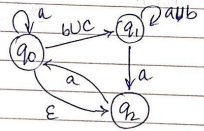
② ADD NEW START STATE (INIT) AND FINAL STATE (F)



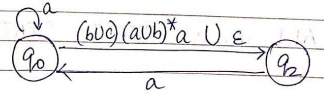
③ RIP q_1 : "IN" STATES : q_0
"OUT" STATES : q_2, F

NEED TO FIX ARROWS (q_0, q_2) AND (q_0, F)

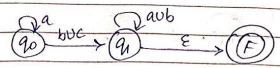
FIXING $q_0 \rightarrow q_2$: OLD :



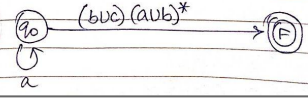
USING SLIDE 6 : NEW :



FIXING $q_0 \rightarrow F$: OLD :

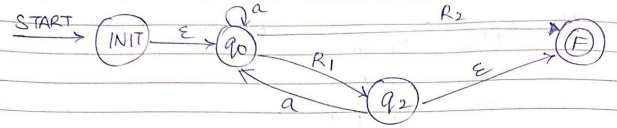


USING SLIDE 6 : NEW :



Example 2 (contd)

∴ AFTER RIPPING q_1, WE GET:



WHERE,

$R_1 = \varepsilon \cup (b \cup c)(a \cup b)^* a$

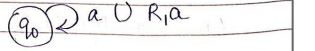
$R_2 = (b \cup c)(a \cup b)^*$

④ RIP q_2 : "IN" STATES : q_0

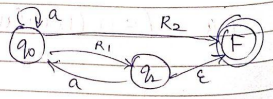
"OUT" STATES : q_0, F.

FIXING : $q_0 \to q_0$: OLD:

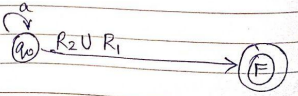
USING SLIDE 6 : NEW :



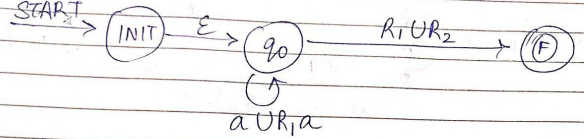
FIXING : $q_0 \to F$: OLD:



USING SLIDE 6 : NEW :



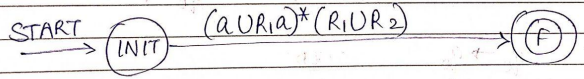
∴ AFTER RIPPING q_2, WE GET:



⑤ RIP q_0 : "IN" STATE : INIT

"OUT" STATE : F

∴ AFTER RIPPING q_0, WE GET:



THE FINAL REGEX IS:

$(a \cup R_1 a)^* (R_1 \cup R_2)$

On SUBSTITUTING VALUES OF R_1, R_2 WE GET:

$$(a \cup (\varepsilon \cup (b \cup c)(a \cup b)^* a)\, a)^* (\varepsilon \cup (b \cup c)(a \cup b)^* a \cup (b \cup c)(a \cup b)^*)$$

Example 3

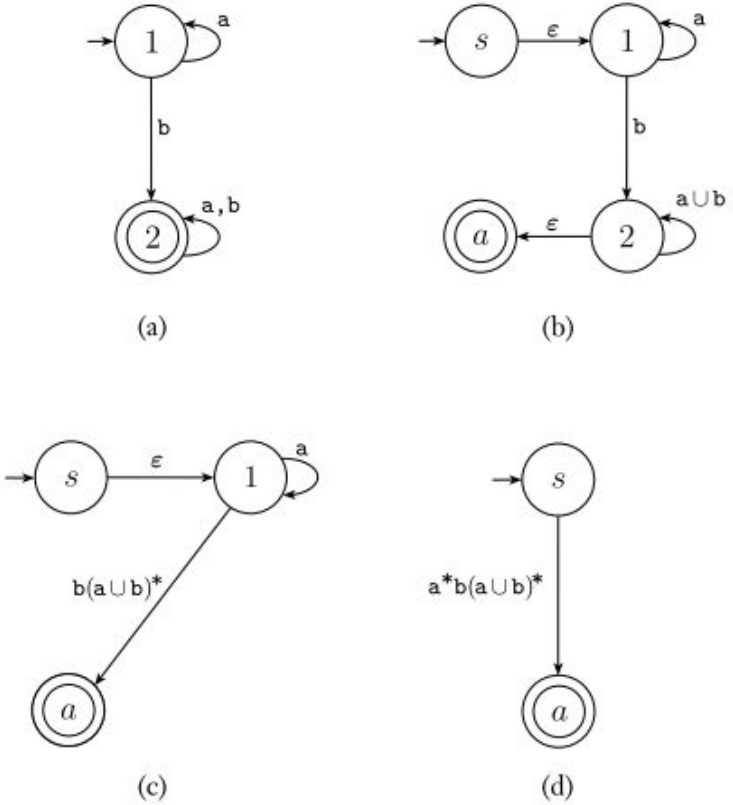
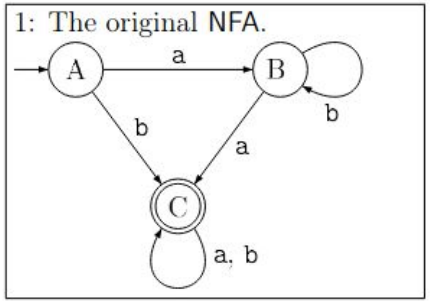
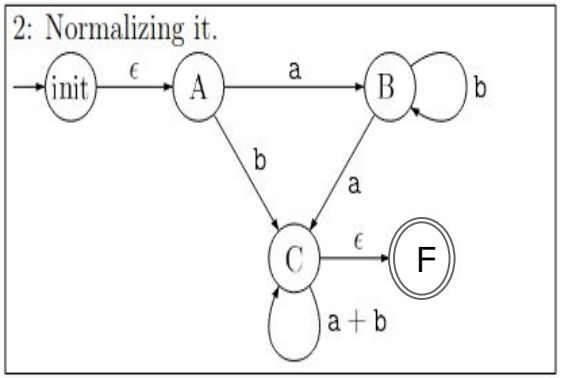


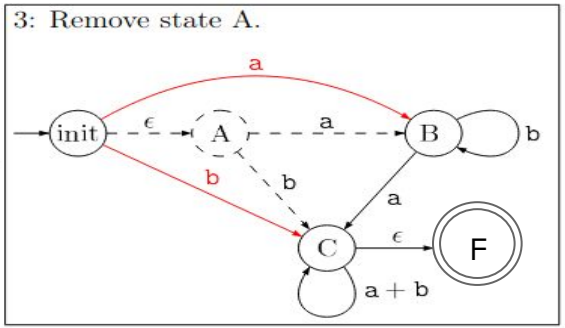
FIGURE 1.67
Converting a two-state DFA to an equivalent regular expression

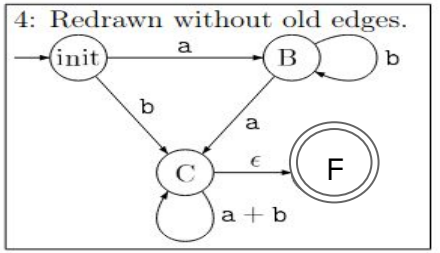
Example 4

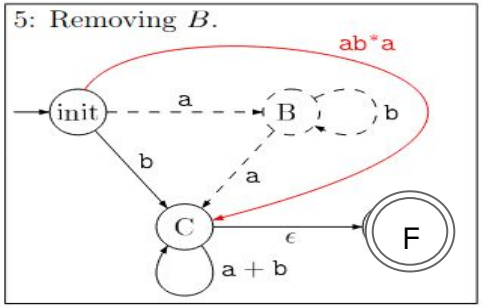


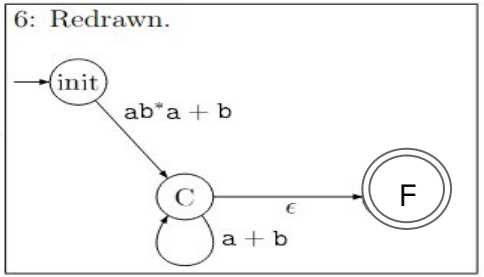
2: Add new START (init) , FINAL (F) state. Notice, self-loop for state C is in regex form.

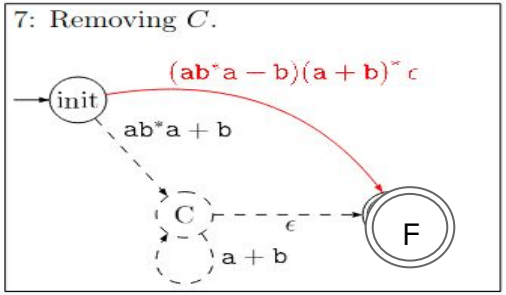


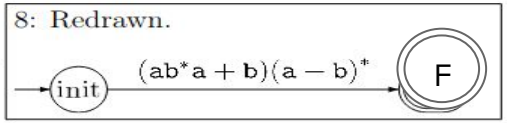












Thus, this automata is equivalent to the regular expression $(ab^*a + b)(a + b)^*$.

Example 5

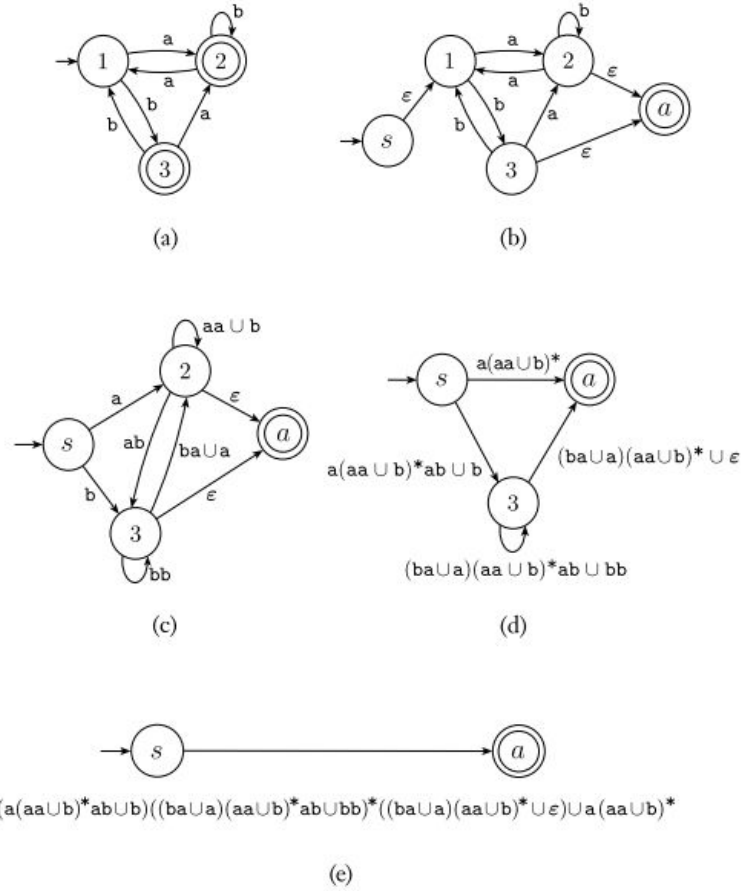


FIGURE 1.69
Converting a three-state DFA to an equivalent regular expression

Alternative algorithms

Some other algorithms that can be used to convert finite automata to a regular expression are:

1. Arden's Theorem
2. Kleene's Algorithm
3. Elimination using Brzozowski

Feel free to explore these algorithms, even though it has not been covered in this presentation.

References / Additional Resources

1. [PDF Book] Introduction to the Theory of Computation by Michael Sipser: https://drive.google.com/file/d/15qeB3VHpsZSRodiJczCnLrlqsK0nHgK_/view?usp=sharing

2. [PDF] Relevant lecture from Illinois : https://courses.engr.illinois.edu/cs373/sp2009/lectures/lect_08.pdf

3. [VIDEO] YouTube tutorials:
   https://www.youtube.com/watch?v=nCsY0LBuE_4
   https://www.youtube.com/watch?v=Gfa3WUFVsJA