

In this note, we show that any DFA can be converted into a regular expression. Our construction would work by allowing regular expressions to be written on the edges of the DFA, and then showing how one can remove states from this generalized automata (getting a new equivalent automata with the fewer states). In the end of this state removal process, we will remain with a generalized automata with a single initial state and a single accepting state, and it would be then easy to convert it into a single regular expression.

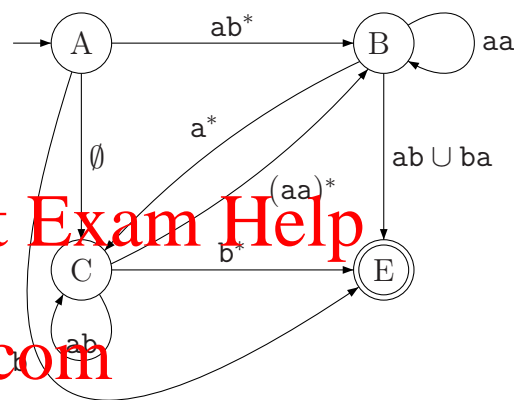
## 1 From NFA to regular expression

### 1.1 NFA— A Generalized NFA

Consider an NFA  $N$  where we allowed to write any regular expression on the edges, and not only just symbols. The automata is allowed to travel on an edge, if it can match a prefix of the unread input, to the regular expression written on the edge. We will refer to such an automata as a *NFA* (*generalized non-deterministic finite automata* [Don't you just love all these shortcuts?]).

Thus, the NFA on the right, accepts the string *abbbbaaba*, since

$A \xrightarrow{abbbb} B \xrightarrow{aa} B \xrightarrow{ba} E.$



To simplify the discussion, we would enforce the following conditions:

- (C1) There are transitions going from the initial state to all other states, and there are no transitions into the initial state.
- (C2) There is a single accept state that has only transitions coming into it (and no outgoing transitions).
- (C3) The accept state is distinct from the initial state.
- (C4) Except for the initial and accepting states, all other states are connected to all other states via a transition. In particular, each state has a transition to itself.

When you can not actually go between two states, a NFA has a transitions labelled with  $\emptyset$ , which will not match any string of input characters. We do not have to draw these transitions explicitly in the state diagrams.

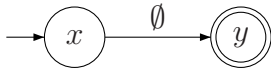
### 1.2 Top-level outline of conversion

We will convert a DFA to a regular expression as follows:

- (A) Convert DFA to a NFA, adding new initial and final states.
- (B) Remove all states one-by-one, until we have only the initial and final states.

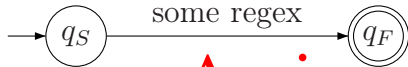
(C) Output regex is the label on the (single) transition left in the NFA. (The word *regex* is just a shortcut for regular expression.)

**Lemma 1.1.** *A DFA  $M$  can be converted into an equivalent NFA  $G$ .*

*Proof:* We can consider  $M$  to be an NFA. Next, we add a special initial state  $q_{\text{init}}$  that is connected to the old initial state via  $\varepsilon$ -transition. Next, we add a special final state  $q_{\text{fin}}$ , such that all the final states of  $M$  are connected to  $q_{\text{fin}}$  via an  $\varepsilon$ -transition. The modified NFA  $M'$  has an initial state and a single final state, such that no transition enters the initial state, and no transition leaves the final state, thus  $M'$  comply with conditions (C1–C3) above. Next, we consider all pair of states  $x, y \in Q(M')$ , and if there is no transition between them, we introduce the transition . The resulting NFA  $G$  from  $M'$  is now compliant also with condition (C4).

It is easy now to verify that  $G$  is equivalent to the original DFA  $M$ . ■

We will remove all the intermediate states from the GNFA, leaving a NFA with only initial and final states, connected by one transition with a (typically complex) label on it. The equivalent regular expression is obvious: the label on the transition.



**Lemma 1.2.** *Given a NFA  $N$  with  $k = 2$  states, one can generate an equivalent regular expression.*

*Proof:* A NFA with only two states (that comply with conditions (C1)–(C4)) have the following form.



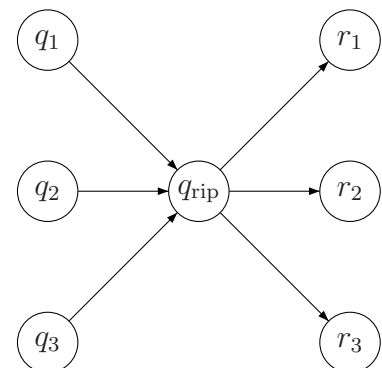
The NFA has a single transition from the initial state to the accepting state, and this transition has the regular expression  $R$  associated with it. Since the initial state and the accepting state do not have self loops, we conclude that  $N$  accepts all words that matches the regular expression  $R$ . Namely,  $L(N) = L(R)$ . ■

### 1.3 Details of ripping out a state

We first describe the construction. Since  $k > 2$ , there is at least one state in  $N$  which is not initial or accepting, and let  $q_{\text{rip}}$  denote this state. We will “rip” this state out of  $N$  and fix the NFA, so that we get a NFA with one less state.

Transition paths going through  $q_{\text{rip}}$  might come from any of a variety of states  $q_1, q_2$ , etc. They might go from  $q_{\text{rip}}$  to any of another set of states  $r_1, r_2$ , etc.

For each pair of states  $q_i$  and  $r_i$ , we need to convert the transition through  $q_{\text{rip}}$  into a direct transition from  $q_i$  to  $r_i$ .



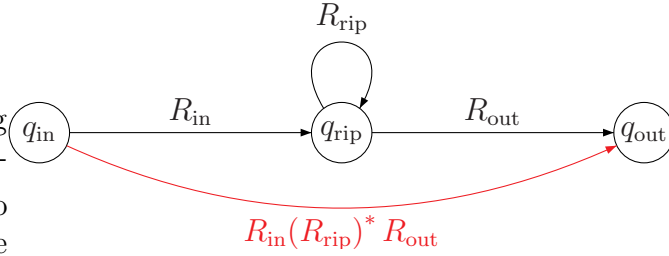
### 1.3.1 Reworking connections for specific triple of states

To understand how this works, let us focus on the connections between  $q_{\text{rip}}$  and two other specific states  $q_{\text{in}}$  and  $q_{\text{out}}$ . Notice that  $q_{\text{in}}$  and  $q_{\text{out}}$  might be the same state, but they both have to be different from  $q_{\text{rip}}$ .

The state  $q_{\text{rip}}$  has a self loop with regular expression  $R_{\text{rip}}$  associated with it. So, consider a fragment of an accepting trace that goes through  $q_{\text{rip}}$ . It transition into  $q_{\text{rip}}$  from a state  $q_{\text{in}}$  with a regular expression  $R_{\text{in}}$  and travels out of  $q_{\text{rip}}$  into state  $q_{\text{out}}$  on an edge with the associated regular expression being  $R_{\text{out}}$ . This trace, corresponds to the regular expression  $R_{\text{in}}$  followed by 0 or more times of traveling on the self loop ( $R_{\text{rip}}$  is used each time we traverse the loop), and then a transition out to  $q_{\text{out}}$  using the regular expression  $R_{\text{out}}$ . As such, we can introduce a direct transition from  $q_{\text{in}}$  to  $q_{\text{out}}$  with the regular expression

$$R = R_{\text{in}}(R_{\text{rip}})^* R_{\text{out}}.$$

Clearly, any fragment of a trace traveling  $q_{\text{in}} \rightarrow q_{\text{rip}} \rightarrow q_{\text{out}}$  can be replaced by the direct transition  $q_{\text{in}} \xrightarrow{R} q_{\text{out}}$ . So, let us do this replacement for any two such stages, we connect them directly via a new transition, so that they no longer need to travel through  $q_{\text{rip}}$ .



Clearly, if we do that for all such pairs, the new automata accepts the same language, but no longer need to use  $q_{\text{rip}}$ . As such, we can just remove  $q_{\text{rip}}$  from the resulting automata. And let  $M'$  denote the resulting automata.

The automata  $M'$  is not quite legal, yet. Indeed, we will have now parallel transitions because of the above process (we might even have parallel self loops). But this is easy to fix: We replace two such parallel transitions  $q_i \xrightarrow{R_1} q_j$  and  $q_i \xrightarrow{R_2} q_j$ , by a single transition

$$q_i \xrightarrow{R_1 + R_2} q_j.$$

As such, for the triple  $q_{\text{in}}, q_{\text{rip}}, q_{\text{out}}$ , if the original label on the direct transition from  $q_{\text{in}}$  to  $q_{\text{out}}$  was originally  $R_{\text{dir}}$ , then the output label for the new transition (that skips  $q_{\text{rip}}$ ) will be

$$R_{\text{dir}} + R_{\text{in}}(R_{\text{rip}})^* R_{\text{out}}. \quad (1)$$

Clearly the new transition, is equivalent to the two transitions it replaces. If we repeat this process for all the parallel transitions, we get a new NFA  $M$  which has  $k - 1$  states, and furthermore it accepts exactly the same language as  $N$ .

## 1.4 Proof of correctness of the ripping process

**Lemma 1.3.** *Given a NFA  $N$  with  $k > 2$  states, one can generate an equivalent NFA  $M$  with  $k - 1$  states.*

*Proof:* Since  $k > 2$ ,  $N$  contains least one state in  $N$  which is not accepting, and let  $q_{\text{rip}}$  denote this state. We will “rip” this state out of  $N$  and fix the NFA, so that we get a NFA with one less state.

For every pair of states  $q_{\text{in}}$  and  $q_{\text{out}}$ , both distinct from  $q_{\text{rip}}$ , we replace the transitions that go through  $q_{\text{rip}}$  with direct transitions from  $q_{\text{in}}$  to  $q_{\text{out}}$ , as described in the previous section.

**Correctness.** Consider an accepting trace  $T$  for  $N$  for a word  $w$ . If  $T$  does not use the state  $q_{\text{rip}}$  then the same trace exactly is an accepting trace for  $M$ . So, assume that it uses  $q_{\text{rip}}$ , in particular, the trace looks like

$$T = \dots q_i \xrightarrow{S_i} q_{\text{rip}} \xrightarrow{\overbrace{S_{i+1} \dots S_{j-1}}^{0 \text{ or more times}}} q_{\text{rip}} \xrightarrow{S_{j-1}} q_j \dots$$

Where  $S_i S_{i+1} \dots, S_j$  is a substring of  $w$ . Clearly,  $S_i \in R_{\text{in}}$ , where  $R_{\text{in}}$  is the regular expression associated with the transition  $q_i \rightarrow q_{\text{rip}}$ . Similarly,  $S_{j-1} \in R_{\text{out}}$ , where  $R_{\text{out}}$  is the regular expression associated with the transition  $q_{\text{rip}} \rightarrow q_j$ . Finally,  $S_{i+1} S_{i+2} \dots S_{j-1} \in (R_{\text{rip}})^*$ , where  $R_{\text{rip}}$  is the regular expression associated with the self loop of  $q_{\text{rip}}$ .

Now, clearly, the string  $S_i S_{i+1} \dots S_j$  matches the regular expression  $R_{\text{in}}(R_{\text{out}})^* R_{\text{out}}$ . in particular, we can replace this portion of the trace of  $T$  by

$$T = \dots q_i \xrightarrow{S_i S_{i+1} \dots S_{j-1} S_j} q_j \dots$$

This transition is using the new transition between  $q_i$  and  $q_j$  introduced by our construction. Repeating this replacement process in  $T$  till all the appearances of  $q_{\text{rip}}$  are removed, results in an accepting trace  $\hat{T}$  of  $M$ . Namely, we proved that any string accepted by  $N$  is also accepted by  $M$ .

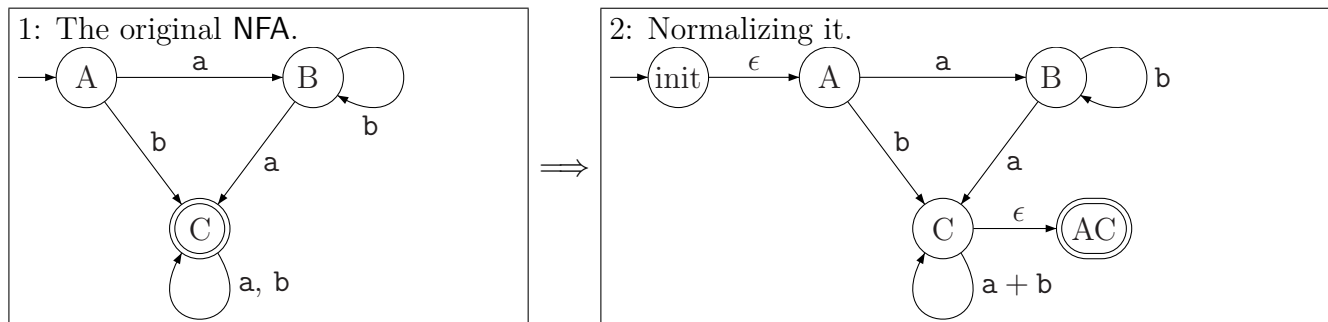
We need also to prove the other direction. Namely, given an accepting trace for  $M$ , we can rewrite it into an equivalent trace of  $N$  which is accepting. This is easy, and done in a similar way to what we did above. Indeed, if a portion of the trace uses a new transition of  $M$  (that does not appear in  $N$ ), we can place it by a fragment of transitions going through  $q_{\text{rip}}$ . In light of the above proof, it is easy and we omit the straightforward but tedious details. ■

**Theorem 1.4.** Any DFA can be translated into an equivalent regular expression.

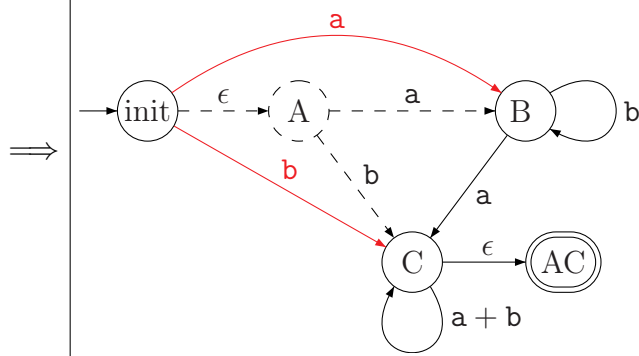
*Proof:* Indeed, convert the DFA into a NFA  $N$ . As long as  $N$  has more than two states, reduce its number of states by removing one of its states using Lemma 1.3. Repeat this process till  $N$  has only two states. Now, we convert this NFA into an equivalent regular expression using Lemma 1.2. ■

## 2 Examples

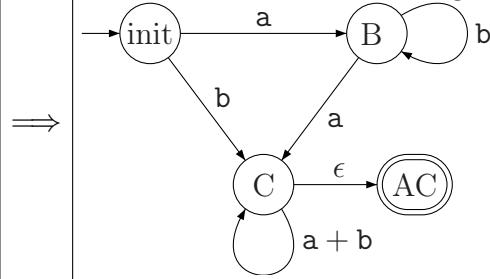
### 2.1 Example: From NFA to regex in 8 easy figures



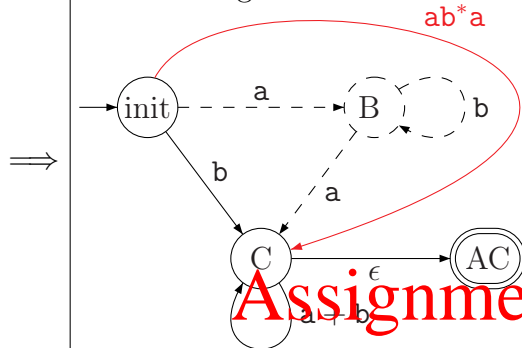
3: Remove state A.



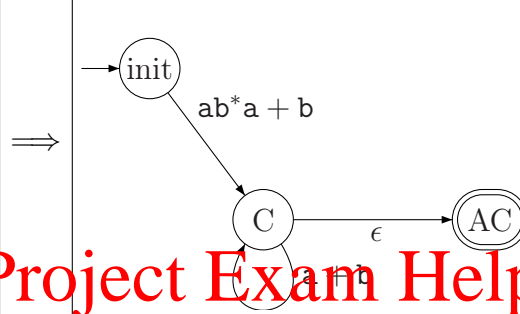
4: Redrawn without old edges.



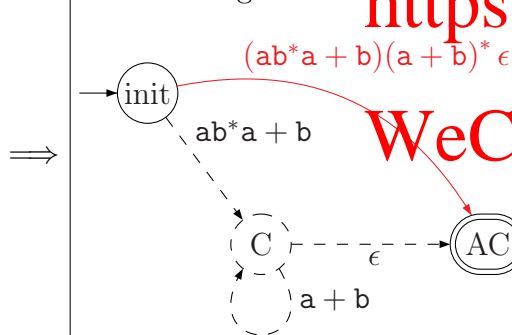
5: Removing B.



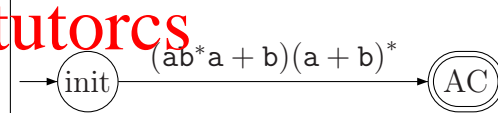
6: Redrawn.



7: Removing C.



8: Redrawn.



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