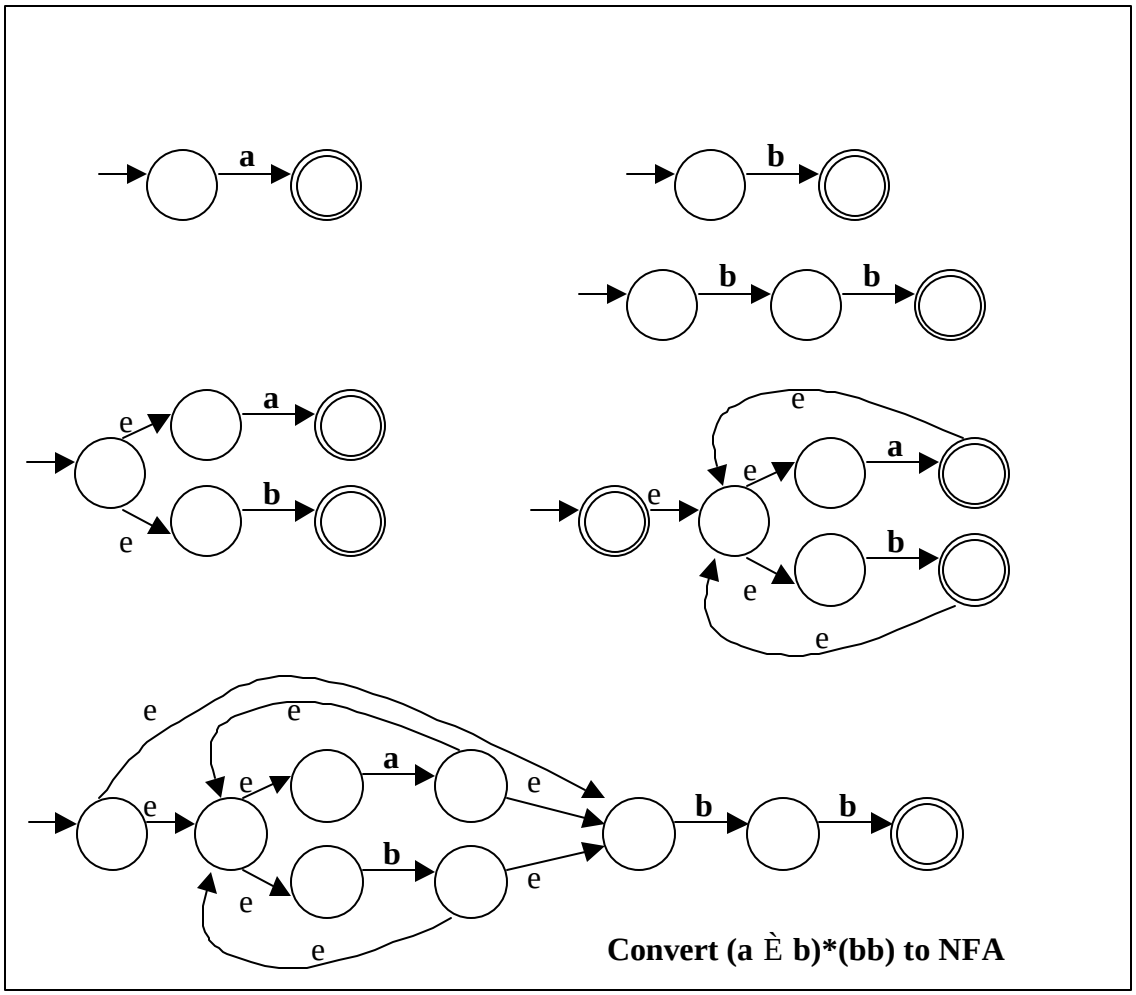
Error! Reference source not found.

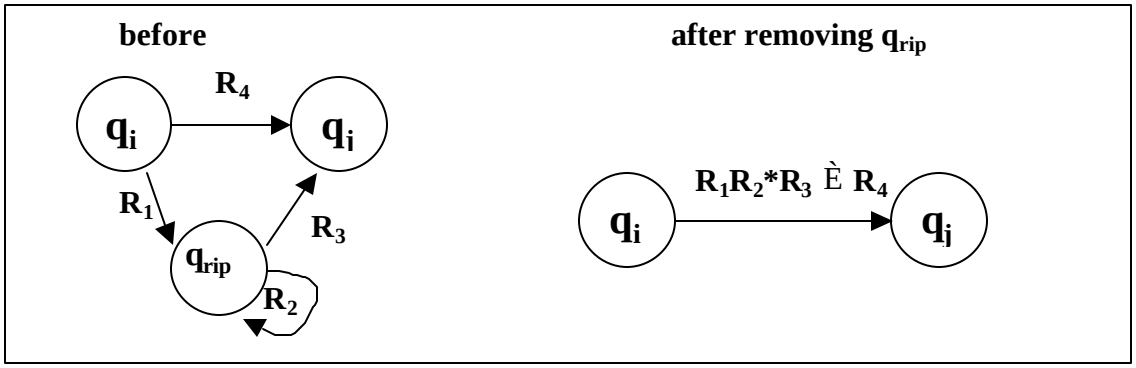
Main topics of the week:

- **Equivalence of regular languages and regular expressions**
- **Construction of NFA from a regular expression**
- **Construction of GNFA from DFA**
- **Reduction of GNFA to a regular expression**
- **Pumping lemma and examples**

Example of constructing an NFA from a regular expression:

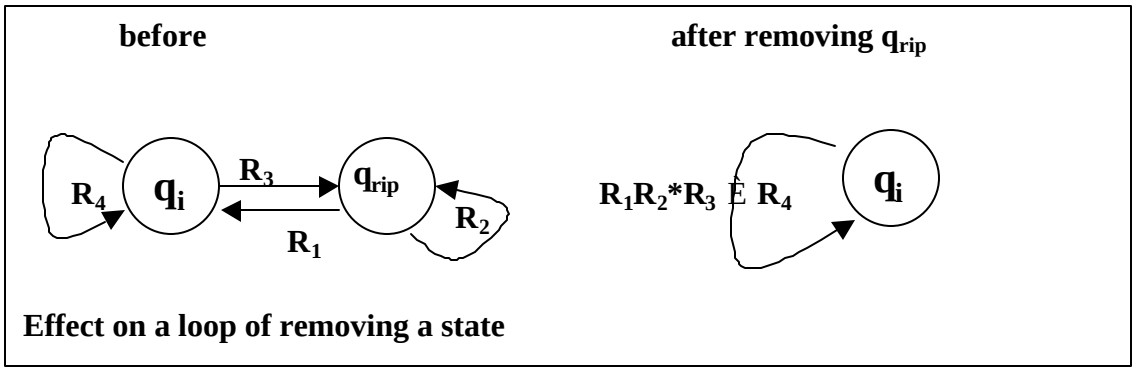


Ripping states out of a GNFA: The pictured technique shows how to rip out a state from a GNFA without affecting the language recognized by the GNFA (at the cost of more complicated RE labels on the remaining state transitions). If we can rip out states one at a time until we are down to just the start and accept state, we will reduce to a very simple GNFA that has one arrow labeled with a regular expression. The following diagram shows this reduction.



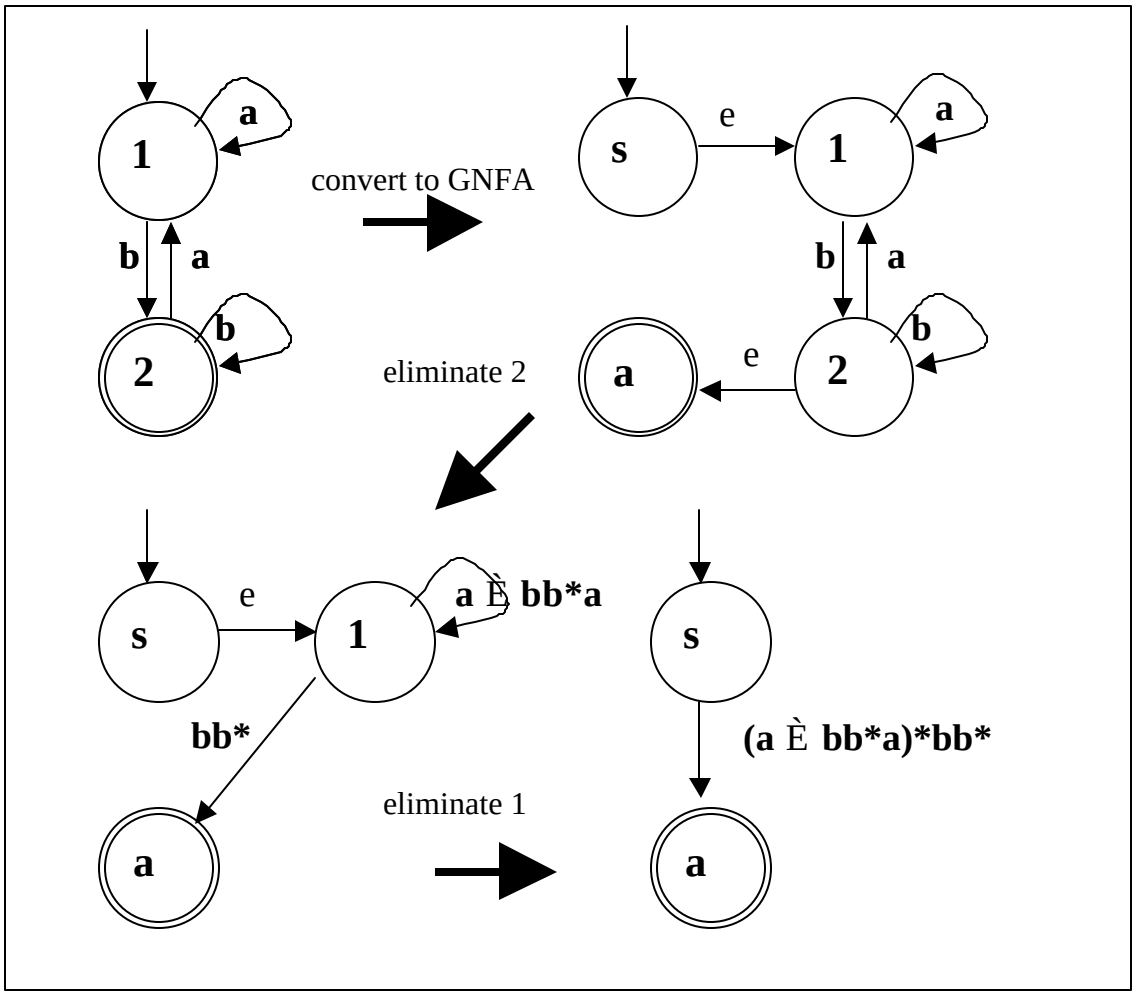
The diagram only shows what happens to the transition from q_i to q_j. What we are doing is enhancing the regular expression that goes between the remaining states with the things we lose when we rip out a state. This includes what gets us into the ripped out state, looping in the ripped out state, and out of the ripped out state. Essentially, we look at what the ripped out state contributes as a possible path between the remaining states, and union it with the original direct path between those two states. Of course, we're showing this in isolation, with only one direction. In a real setting there would be both directions as well as all the other states the ripped out state is connected to. But in terms of re-labeling arrows, what we show is exactly what happens for all the affected arrows.

One special case is worth considering: when q_i and q_j are the same state. In the following diagram, we show what this looks like. Here, the R_4 label from q_i to q_j is just the loop on q_i, and that is the arrow affected by removing q_{rip}.



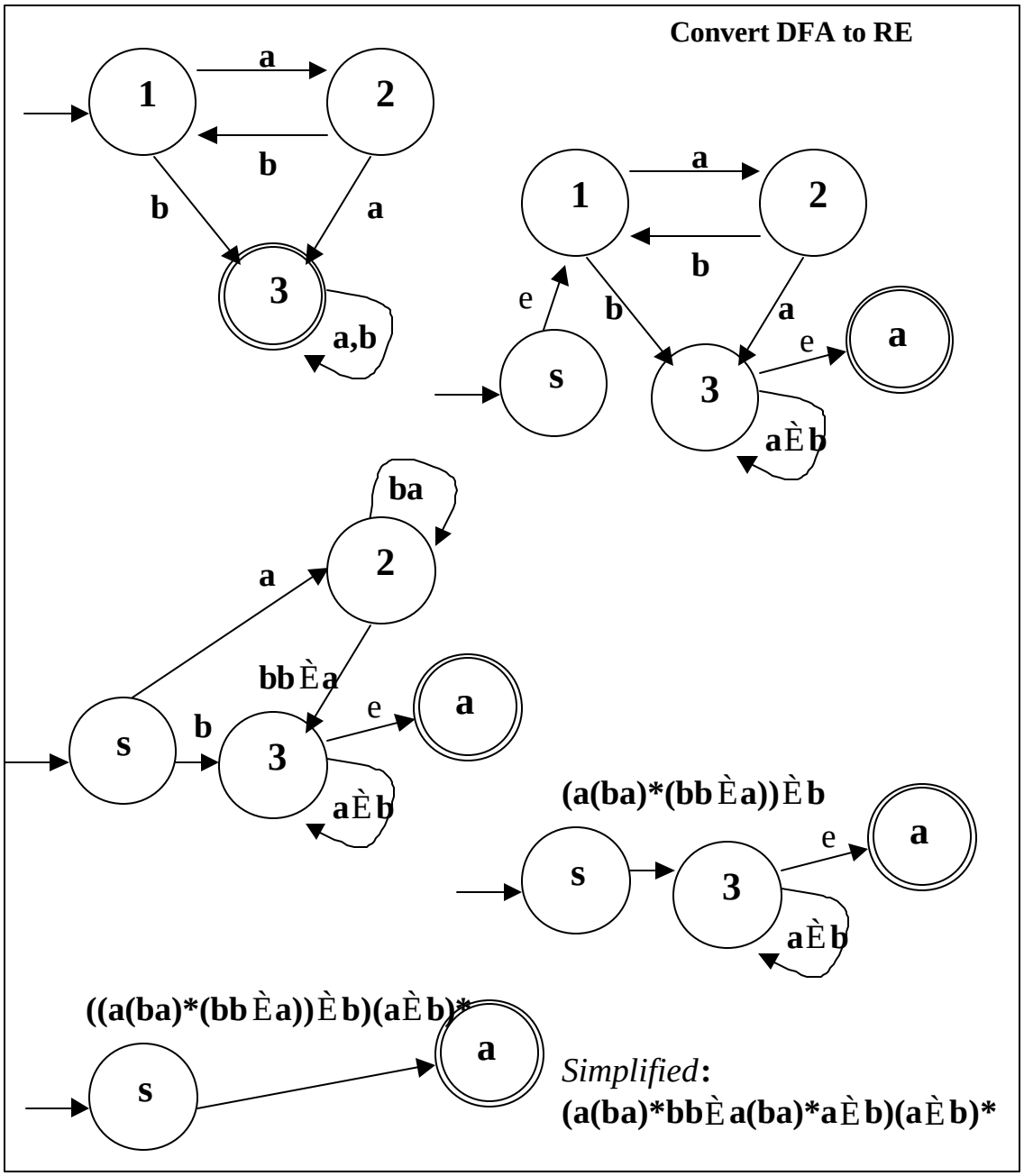
Error! Reference source not found.

Example of reduction for DFA with two states. We first convert the DFA to a GNFA (four states), then reduce the states to three and then to two to end up with the RE equivalent to the DFA. The first conversion to a GNFA is by adding start and accept states and ε transitions in and out of them to/from the original start/accept states. We also combine arrows with union and add arrows with ∅. Here we won't bother with the ∅ arrows, but just remember that a missing arrow is labeled ∅. Also we don't have any multiple arrows to combine with union. In our four state GNFA, we go about eliminating states. First we eliminate state 2, and since we could get from 1 to accept via 2, we use our rule to compose the regular expression of the arrows from 1 to 2 to 2 to accept, getting bb*ε. We suppress the ε as we have seen earlier is an identity. Likewise, we are really taking the union of this with the arrow directly from 1 to accept (which is labeled ∅) and again by our identity, we can suppress this part. When we eliminate 2, we are also collapsing a path from 1 back to itself, so we use the rule in the degenerate case of $q_i=q_j$ to adjust the loop on 1 to be a ∪ bb*a. Now we go about eliminating 1 in the same way and add ε(a ∪ bb*a)* to our composition, with this finally resulting in (a ∪ bb*a)*bb*. Of course, we can also look at this DFA and see that it just describes all strings ending with b, so also would be determined by the regular expression (a ∪ b)*b.



Error! Reference source not found.

Example of reduction starting with three state DFA. We go through the reductions and end up with the regular expression **((a(ba)*(bbÈa))Èb)(aÈb)*** which can be simplified to **(a(ba)*bbÈa(ba)*aÈb)(aÈb)*.** If you think about this, you see that it describes the language where if the string begins with a, then the a must be followed by two b's or another a, with zero or more pairs of ba in between, i.e., the string begins with a(ba)*a or a(ba)*bb, or else the string begins with b.



Pumping Lemma Examples

Consider the language $B = \{0^n1^n \mid n \geq 0\}$ that we intuitively feel is not regular. Suppose that B is regular. Then by the pumping lemma, there is a pumping length p. Consider the string 0^p1^p, and let x, y, and z be the substrings guaranteed by the pumping lemma. It is certainly true that y must consist of just zeroes, or just ones, or both. If y consists of just zeroes, then the string xyyz will have have more zeroes than ones since the y adds only more zeroes, and x and z are fixed. Likewise, if y consists of just ones, then the same reasoning shows that xyyz has more ones than zeroes. So we know that y cannot be all zeroes or all ones. So y must have some of each. However, if we again look at the string xyyz, we would see that the zeroes of the second y occurrence would occur after the ones in the first y occurrence, again a contradiction. Since all of the possibilities for y lead to a contradiction, we conclude that there can be no pumping length, hence B is not a regular language.

This proof can be simplified by using condition 3 from the pumping lemma: that $|xy| \leq p$. Using this, we can assert that y must consist entirely of zeroes and thus not have to argue all the separate cases.

The art of using the pumping lemma is being clever about the choice of the string that produces a contradiction. Above it was pretty easy since any string of length p would have worked, but that is not always the case.

For another example, let P be the language of all palindromes, i.e., strings that are the same when read in the reverse direction. If P is regular, then we have a pumping length p. Consider the string a^pba^p. Certainly this is a palindrome. In our decomposition xyz, we are guaranteed that the length of xy does not exceed p. Thus, xy must consist just of a's and in particular this will be true of y, say $y=a^k$, where $k \geq 1$. But then xz would have the form $a^{p-k}ba^p$, and this is obviously not a palindrome, so P is not regular.

Here's another example that uses a little more reasoning about lengths. Let $T = \{0^k \mid k = 2^n \text{ for some } n \geq 0\}$. That is, T is the set of all strings of zeroes whose length is a power of 2. If it is regular, then the pumping lemma applies. Let s be a string of length $m \geq p$, and $s=xyz$. We know that $y = 0^n$ for some n and $m = 2^k$ for some $k \geq 0$. Note that xz will have length $2^k - n$, which must also be a power of 2, say $2^k - n = 2^i$ for some $i < k$. Similarly, if we look at xyyz, we see that $2^k + n = 2^j$ for some $j > i$. Adding these together gives $2^{k+1} = 2^i + 2^j = 2^i (1 + 2^{j-i})$. Dividing out the 2^i leaves us with the left hand side a power of two, hence even, and the right hand side as 1 plus a power of two (power at least one), which is odd. Thus T is not regular.