COMP 330 Fall 2023
Supplementary Note
Lecture 7

Cesare Spinoso-Di Piano

Last updated: September 24, 2023

This supplementary note provides a more detailed presentation of the FA to regular expression conversion algorithm as well as slightly more complicated examples/illustrations than the one(s) seen in class.

1 FA to RegExp conversion algorithm

Recall from the Lecture 7 notes the "high-level" procedure I gave to convert any FA to a regular expression. This procedure included two major steps. The first one is the conversion of the FA to a GFA and the second is the "shrinking" of this GFA.

1.1 Conversion to a GFA

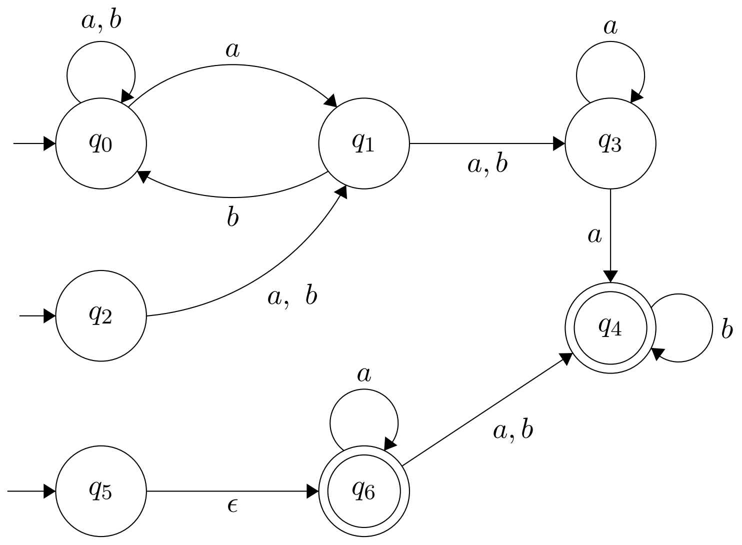
The first step of the conversion algorithm is, given some input FA N, to convert N into a generalized finite automaton (GFA) N'. A GFA must have the following properties:

1. It must have a single start state with only outgoing edges.
2. It must have a single accept state with only incoming edges.
3. The labels to its transitions are of type "regular expression". If the label of an edge from state p to q in a GFA N' is r, then this means that, given any string which matches with the pattern in r, N' can read that entire string and transition from p to q.

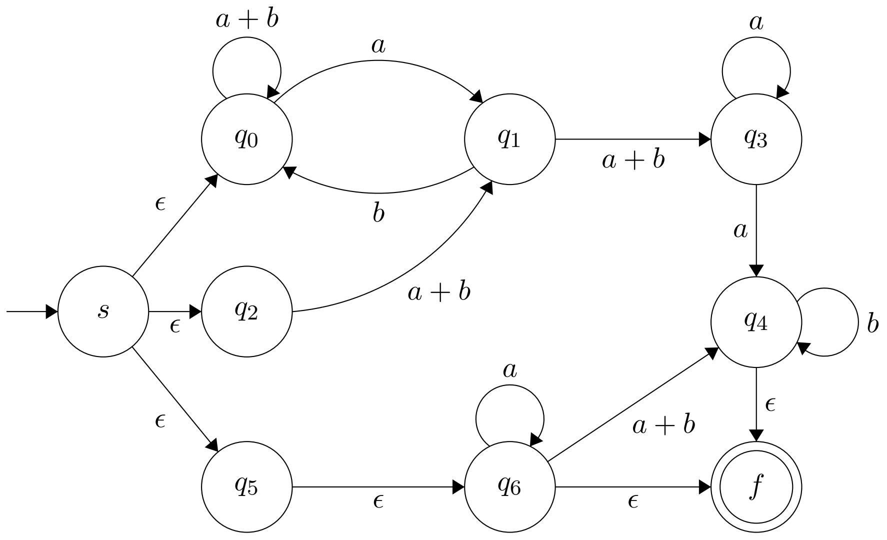
I will not provide a complete formalization of GFA (i.e., a tuple-like definition), but it shouldn't be too difficult for you to see how you could do so. In particular, the major difference would be in the way you define the transition function.

We saw that it is relatively easy to convert any FA to a GFA. In order to meet conditions (a) and (b), we can leverage the convenience of ϵ-transitions (which in the case of GFA are really transitions labeled with the *atomic regular expression* that looks like the empty string). For example, consider the following NFA+ϵ[1]

[1] Please don't ask me what this machine accepts. I have no idea. I suppose trying to figure that out could be a good exercise, but don't waste too much time on it.



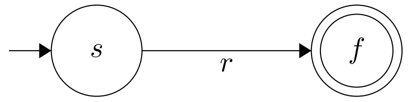
Then the equivalent GFA is



Where state s is the unique start state with only outgoing edges and f is the unique accept state with only incoming edges.

1.2 Shrinking of the GFA

The second step of the conversion algorithm is to "shrink" the GFA, N', by ripping all intermediate states (i.e., all states that are not the start and accept state). The final GFA, N'', will look like



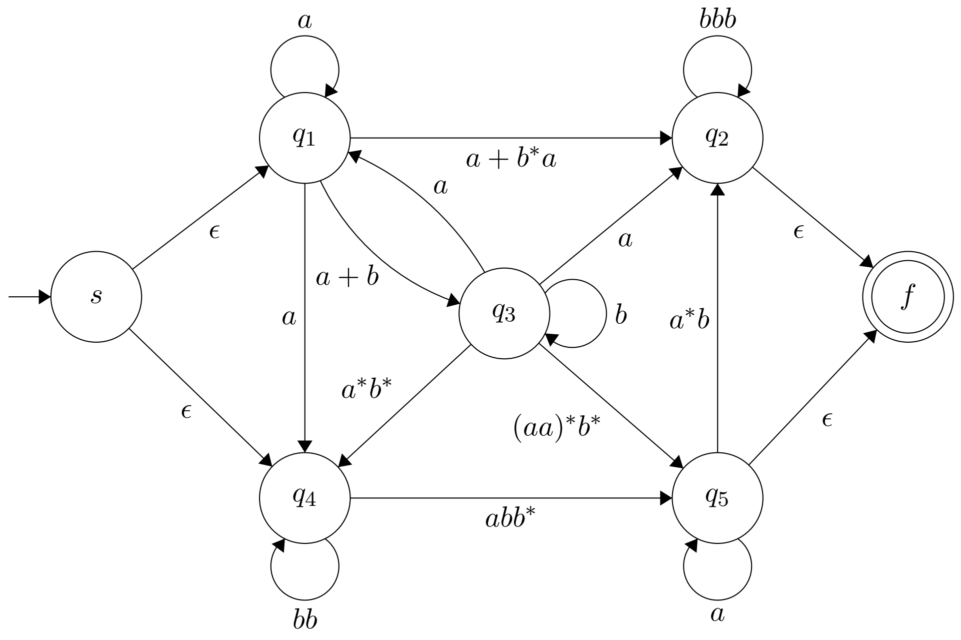
At which point, it is clear that the equivalent regular expression for N'' is r since $L(r) = L(N'')$. However, if we are careful about how we rip out states and stitch the machine back together, then we will have found an equivalent regular expression for N' as well. Here is a more explicit procedure of how this rip/stitch process should be done. In this procedure, we let r_{ab} be the regular expression which labels the transition from q_a to q_b.

1. Given a GFA N', if N' has 2 states, return N'. Otherwise, pick some intermediate state and call it q_k. This will be the state we rip out of the machine.
2. Copy all of the states and transitions of N' except for the state q_k and any of its incoming/outgoing edges. Call this copied machine N''.
3. For every pair of states q_i, q_j from N'' (q_i, q_j may be the same state), do the following:
   (a) Check whether there is a path from q_i to q_j labeled $q_i \xrightarrow[r_{ik}]{} q_k \xrightarrow[r_{kk}]{} q_k \xrightarrow[r_{kj}]{} q_j$ in N'. If yes, let $r' := r_{ik}r_{kk}^*r_{kj}$. Otherwise, check whether there is path $q_i \xrightarrow[r_{ik}]{} q_k \xrightarrow[r_{kj}]{} q_j$ in N'. If yes, let $r' := r_{ik}r_{kj}$. Otherwise, let $r' := \emptyset$.
   (b) If $r' \equiv \emptyset$, do nothing. Otherwise, if $r' \not\equiv \emptyset$, and there is an edge labeled r_{ij} from q_i to q_j in N' then the edge in N'' will be labeled $r_{ij} + r'$. Otherwise, create an edge from q_i to q_j and label it r' in N''.
4. Repeat step 1. on N''.

It should be clear to you why all of the information in the GFA is preserved. At any particular step of this procedure, if, when reading w, the GFA could transition from q_i to q_j through q_k, then it will be equally as able to do so in the machine without q_k by using the edge from q_i to q_j. This transition is possible because the pattern w matched with so N' could transition from q_i to q_j through q_k has been added to the pattern of strings that allow N'' to transition directly from q_i to q_j.

Let's illustrate *one step* of this procedure for the following GFA[2]

[2]I will never *ever* ask you to convert such an FA in a midterm or a final. That would be diabolical. Knowing about this algorithm and its general idea is sufficient.



Suppose we rip out state q_3. We must stitch the machine back together such that no information is lost. I will go through a few pairs of states that will receive additions to their transitions:

$q_1 \to q_1$ The machine can go from q_1 back to q_1 either by reading a string with any number of a's (using its self loop) *or* by reading a string which matches the pattern $(a + b)b^*a$ and transitioning to q_3 and back to q_1. Thus, the label for the edge from q_1 to q_1 in the new GFA will be $a + (a + b)b^*a$.

$q_1 \to q_4$ The machine can go from q_1 to q_4 either by reading the string a *or* by reading a string which matches the pattern $(a + b)b^*a^*b^*$ and transitioning to q_3 and then to q_4. Thus, the label for the edge from q_1 to q_4 in the new GFA will be $a + (a + b)b^*a^*b^*$.

$q_1 \to q_5$ The machine isn't able to read a string and directly transition from q_1 to q_5. Instead, it can read a string which matches the pattern $(a + b)b^*(aa)^*b^*$ and transition through q_3 to q_5. Thus, in the new GFA, a new edge from q_1 to q_5 will be created and will have label $(a + b)b^*(aa)^*b^*$.

I leave the rest of the pairs for you to try out to make sure you've understood the algorithm.