Regular Languages

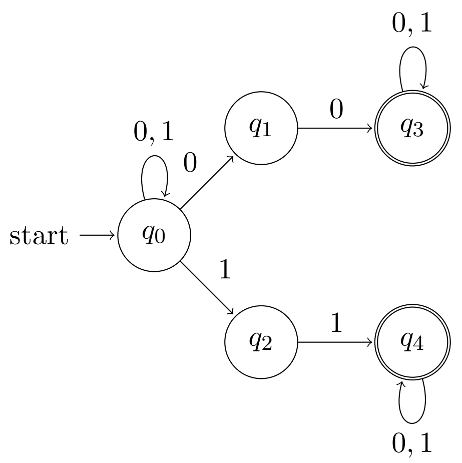
Theorem 1. *Let L be a language. The following statements are equivalent:*

1. $L = L(A)$ *for some NFA* A
2. $L = L(A')$ *for some DFA* A'
3. $L = L(R)$ *for some regexp* R

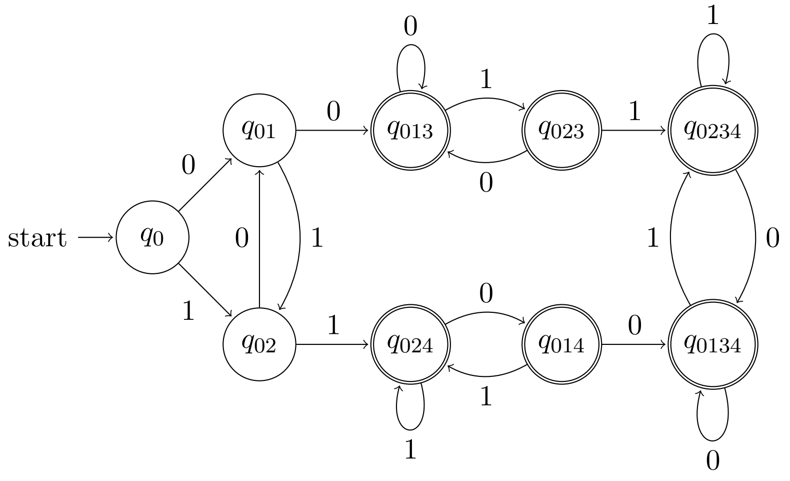
Proof. We are not going to prove this theorem. However, we are going to talk about the main ideas of the proof. You can look at Sections 7.4.2 and Sections 7.6 in the textbook for a foraml treatment of this theorem. To prove the theorem we will use a chain of implications.

$1 \Rightarrow 2$*:* For any NFA A, create DFA A' using subset construction. This ensures that computation of A' simply tracks computation of A, so $L(A') = L(A)$.

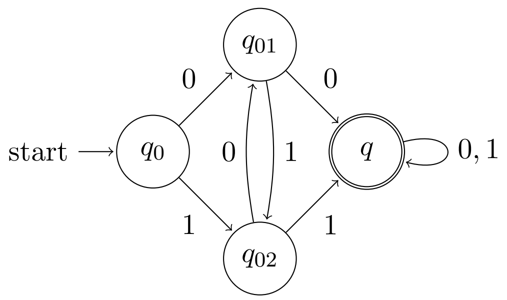
Example 1. *Consider the following NFA:*



The corresponding DFA is:

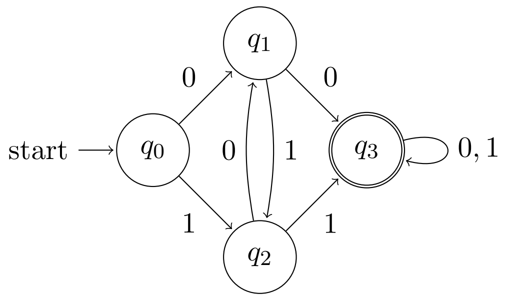


Note that all accepting states above trap in the sense that once the string is accepted, no input symbol can make the DFA go back to rejecting state. So the DFA can be simplified as follows:

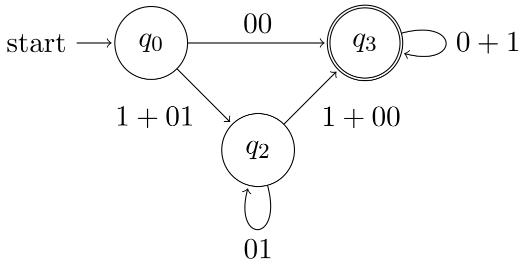


$2 \Rightarrow 3$*:* We can find the regular expression that describes a DFA using **state elimination** construction – eliminate intermediary states and replace transition labels with more general RE's. More precisely, for each accepting state q, eliminate all states except q and initial state to generate RE R_q (representing pattern for all strings accepted at state q). RE for DFA consists of union of RE's for all accepting states. As construction proceeds, labels on transitions become more elaborate RE's (starting from single symbols in original DFA).

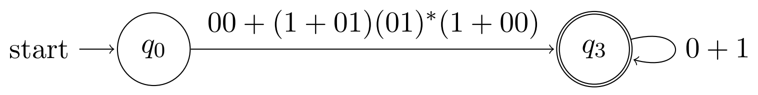
Example 2. *Consider the following DFA:*



Let's eliminate q_1*:*



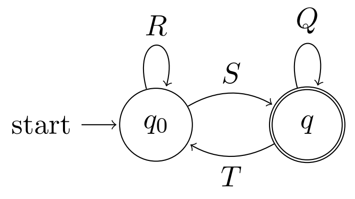
now, eliminate q_2:



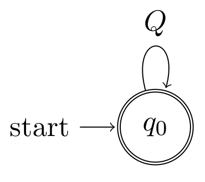
Therefore, $R_{q_3} = (00+(1+01)(01)^*(1+00))(0+1)^*$. *Since q_3 was the only accepting state, this RE is equivalent to DFA.*

In general, you can follow the following instructions to derive the RE corresponding to the DFA:

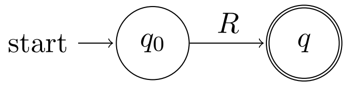
- How to eliminate state q:
  - let $s_1, \cdots, s_m$ be all states that have a transition into q, with labels $S_1, ..., S_m$
  - let $t_1, \cdots, t_n$ be all states that have a transition from q, with labels $T_1, ..., T_n$ (note: it is quite possible to have $s_i = t_j$ for some i, j
  - for each $1 \le i \le m$, $1 \le j \le n$, let $R_{i,j}$ be label of transition from s_i to t_j ($R_{i,j} = \{\}$ if no transition)
  - let Q be label of loop on state q ($Q = \{\}$ if no loop)
  - remove q from FSA and for each $1 \le i \le m$, $1 \le j \le n$, replace label $R_{i,j}$ with $R_{i,j} + S_iQ^*T_j$
  - special cases:
    - ∗ if $R_{i,j} = \{\}$, simplifies to: $s_i \overbrace{\to}^{S_iQ^*T_j} t_j$
    - ∗ if $Q = \{\}$, $S_iQ^*T_j$, simplifies to: S_iT_j
- How to write down R_q for accepting state q:
  - once all states removed except initial state q_0 and final state q, only four possible transitions remain:



  - $R_q = R^*S(Q + TR^*S)^*$
  - special case: q_0 accepting leaves only one state with $R_{q_0} = Q^*$

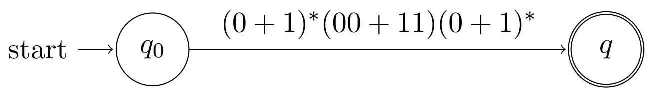


$3 \Rightarrow 1$*:* start with the following NFA

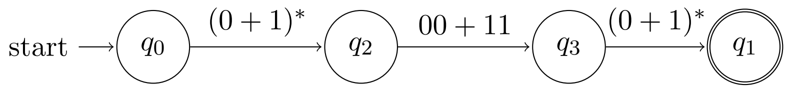


then break up R into component pieces, adding states and transitions as necessary, until each transition labelled by single symbols.

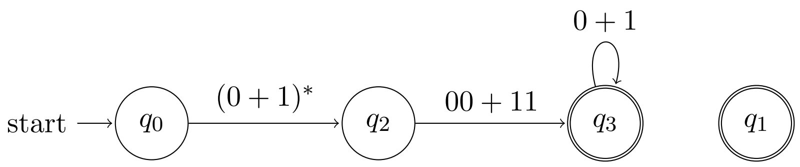
Example 3. *Let* $R = (0+1)^*(00+11)(0+1)^*$*. so start with:*



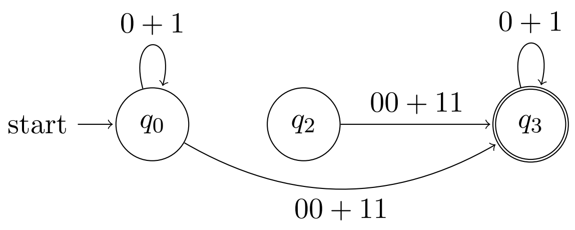
Deal with top-level operators (concatenation) to get:



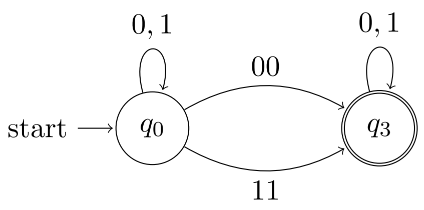
No transition out of q_1 *so deal with last transition next:*



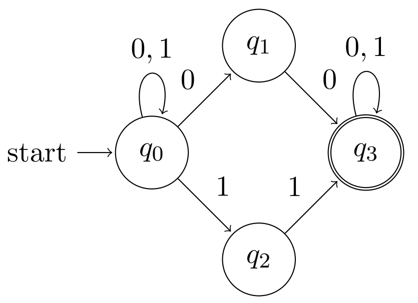
Remove q_1 *(no incoming transition) and deal with first transition:*



Remove q_2 *(no incoming transition), replace '+' with multiple transitions:*



Deal with both concatenations:



There is another way to handle this by introducing ϵ-transitions: transitions labelled by ϵ, that can be followed without processing any input symbol. This introduces another form of non-determinism into the NFAs, but the subset construction can be adjusted to account for it. And it makes the construction of NFAs from REs much easier to describe– but resulting NFAs much more cumbersome (See the textbooks for details.)

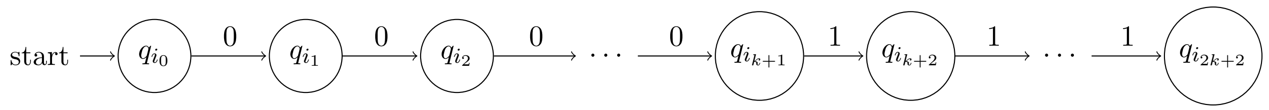
□

Non-regular languages

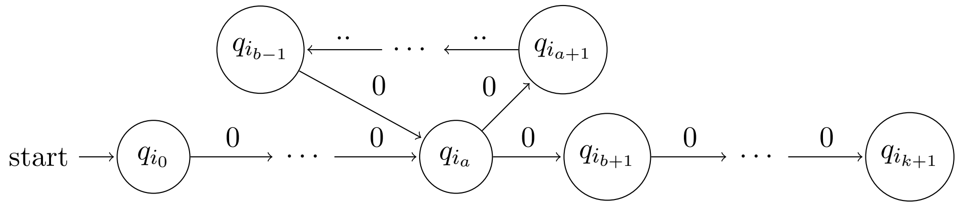
As we pointed out in previous lectures, FSA have fixed, finite memory (states) and hence cannot remember unlimited information about strings.

Example 4. $L = \{0^n 1^n : n \in N\} = \{\epsilon, 01, 0011, 000111, \cdots\}$ *is not regular.*

Proof. For a contradiction, suppose that L is regular. Then, there is a DFA A such that $L(A) = L$. Let k be the number of states in A (so A's states are $\{q_0, q_1, \cdots, q_{k-1}\}$, and consider the behaviour of A on input string $0^{k+1}1^{k+1}$:



where $q_{i_0}, q_{i_1}, \cdots, q_{i_{2k+2}}$ are states of A and $q_{i_{2k+2}}$ is acceping. Since A contains only k states, some state of A must be repeated among $q_{i_0}, \cdots, q_{i_{k+1}}$ – there must be some $a < b \leq k+1$ such that $i_a = i_b$. Schematically, the sequence of states that A goes through on string 0^{k+1} looks like this:



But then, the behaviour of A on input string $0^{k+1+(b-a)}1^{k+1}$ will be the same as on input $0^{k+1}1^{k+1}$, i.e., A accepts some strings that are not in L! This contradicts our assumption that $L(A) = L$, so there can be no such DFA, i.e., L is not regular. □

Context-free grammars are more powerful ways to describe sets of strings and are used to represent many natural-language constructs– in particular, all modern programming languages. Set L in the previous example can be described by such grammars. I will leave a more thorough discussion of them to CSC463 and CSC448.