1) Let N be an arbitrary NFA $= (Q, \Sigma, \delta, q_0, F)$.
We will show how to create another NFA N' with only one accept state such that $L(N) = L(N')$.

$N' = (Q', \Sigma, \delta', q_0, \{q_{final}\})$ with

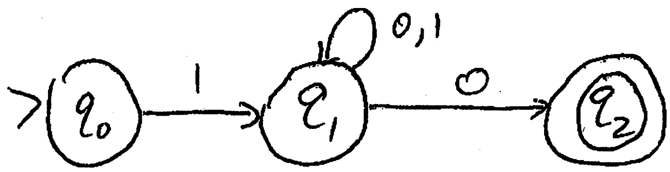
$Q' = Q \cup \{q_{final}\}$

$\delta' = \delta \cup \{ (q_i, \varepsilon) \rightarrow q_{final} \mid q_i \in F \}$

In essence, we add one new state to N, q_{final}, and make it an accept state. Then we add epsilon transitions from each of the original final states in N to q_{final} and leave this as the only accept state.

Now, we show $L(N) = L(N')$. Consider an arbitrary string that is accepted by N. There must exist a path for it to an accept state in N. From there, just take the epsilon transition to q_{final}. Thus N' also accepts this string. Next consider any string accepted by N'. To get to its accept state, the string must have come directly from a state $q_i \in F$ on an epsilon ~~st~~ transition, meaning that N accepts the string. Thus the languages are equal as desired.

2) L_1:

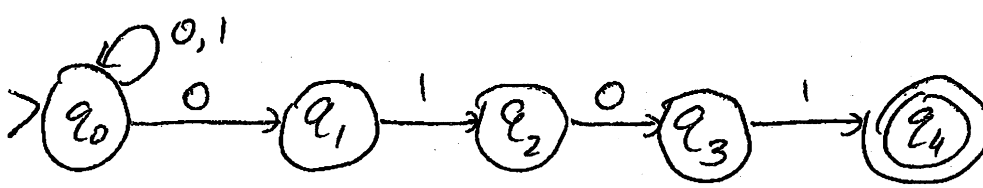


q_1: all strings that start with 1.

q_2: all strings that start with 1 <u>and</u> end in 0.

Designing an NFA for L_1 is a bit easier than a DFA since we don't need to define $\delta(q_0, 0)$ and $\delta(q_2, x)$, $x \in \{0,1\}$.

L_2:



q_1: ends in 0, q_2: ends in 01

q_3: ends in 010, q_4: ends in 0101.

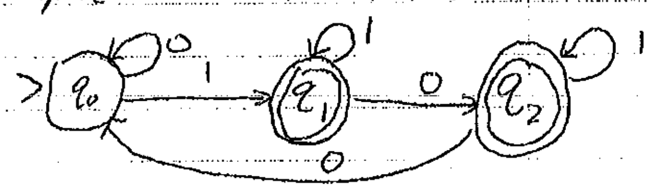
Non-determinism helps here greatly by allowing for us to "guess" the last 4 digits of the string.

4) Let D be a DFA that accepts A. We will use D to create an NFA, ~~A, that~~ N, that accepts A^R. Make N's accept state equal to D's start state. For each transition in D of the form $\delta(q_i, a) = q_j$ create a transition in N of the form $\delta(q_j, a) = q_i$. This "reverses" each transition. Finally, add a new start state to N, q_0'. This state will have ~~ε~~' transitions of the form $\delta(q_0', \varepsilon) = q_i$ for each $q_i \in F$, where F is the set of final states in D. (Note: the only final state in N will be the old start state of D, so none of the $q_i \in F$ in D will be accept states in N, unless $q_0 \in F$)

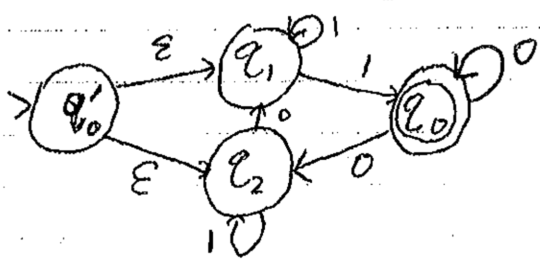
The rationale behind this transformation is to allow each string from the original language to be read in backwards.

Here is an example of the transformation:

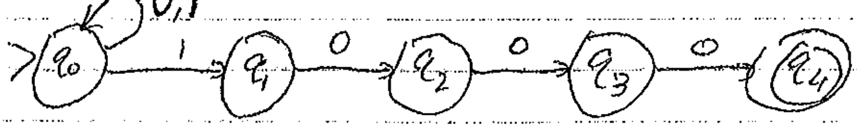
Original DFA for A:



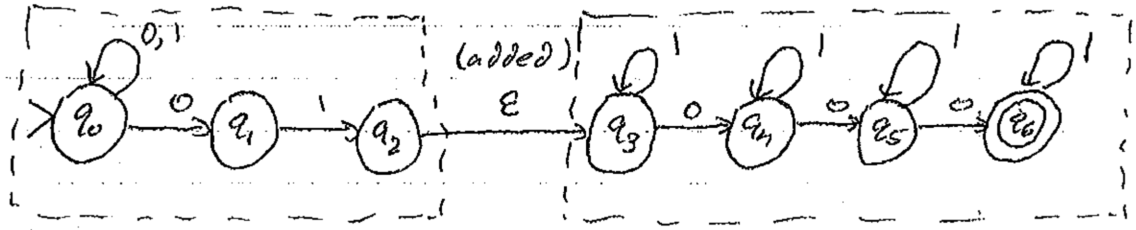
NFA for A^R:



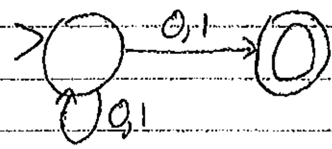
Any sequence of states followed in the original DFA can be "reversed" in the NFA, after the appropriate ε transition is taken.

4) 

We assume that all strings in L must have at least one 1.

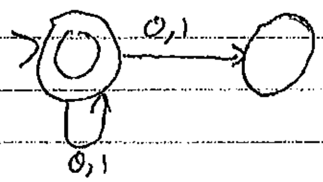
5) 

6) Consider the following NFA:



The language of this NFA is $\Sigma^* - \varepsilon$.

Using Tommy's transformation, the NFA for the language $\overline{L}$ would be



But, this NFA accepts Σ^*, ~~we~~ which is NOT $\overline{\Sigma^* - \varepsilon}$. Specifically the string 1 is accepted by both machines, proving that the second does NOT accept the complement language of the first.

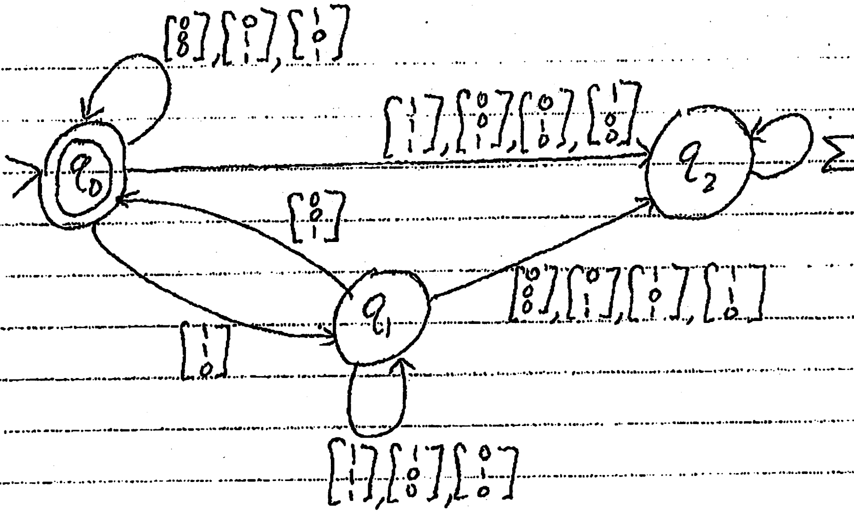
7) We will show that B is regular by creating a DFA that accepts B^R. Our DFA will have 3 states representing these points in our computation:

q_0: What we have read in so far adds up correctly.

q_1: What we have read in matches all the answer bits, BUT we have a carry bit we are expecting in the next character to read in.

q_2: There is already a mistake in the answer bits. Nothing in this state can be accepted, regardless of what is read in, in the future.

7 (cont.)



Explanation of transitions:

$q_0 \to q_0$: In all 3 of these, the first two bits add to the third, maintaining equality

$q_0 \to q_1$: The parity of the bottom bit is the same as the sum of the top two, but a carry bit is produced, so the current string is no longer valid.

$q_0 \to q_2$: The parity of the answer bit is incorrect. All strings with this prefix are NOT in the designated language.

$q_1 \to q_0$: This fixes the previous ~~pari~~ carry bit problem.

$q_1 \to q_1$: The carry bit has cascaded to the next bit.

$q_1 \to q_2$: The current bit is now incorrect, so all strings with this prefix are NOT in the language.