COMS 3261 Handout 6B:
NFA and Regex Practice Practice Solutions

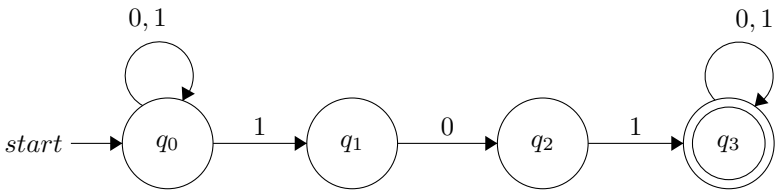
Ziheng Huang and Kiru Arjunan
(Credits to Cindy Wang, TA 2018)

Fall 2024

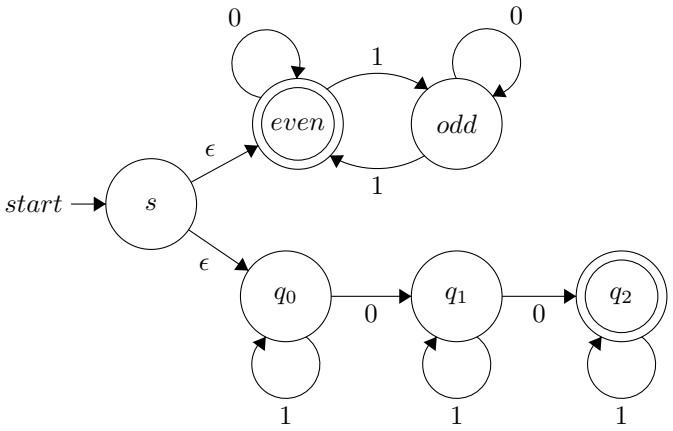
1 NFA

Problem 1. Draw an NFA that recognizes:

(a) All strings that contain 101.



(b) $L = \{w \in \{0,1\}^* \mid w \text{ has exactly two 0's or an even number of 1's}\}$



Problem 2.

(a) What is the language recognized by this NFA?

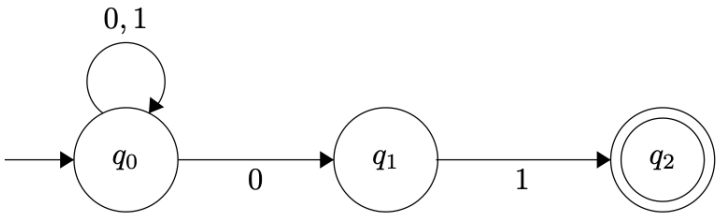
The state q_0 is not accepting. The language recognized by this NFA is $\{\}$.

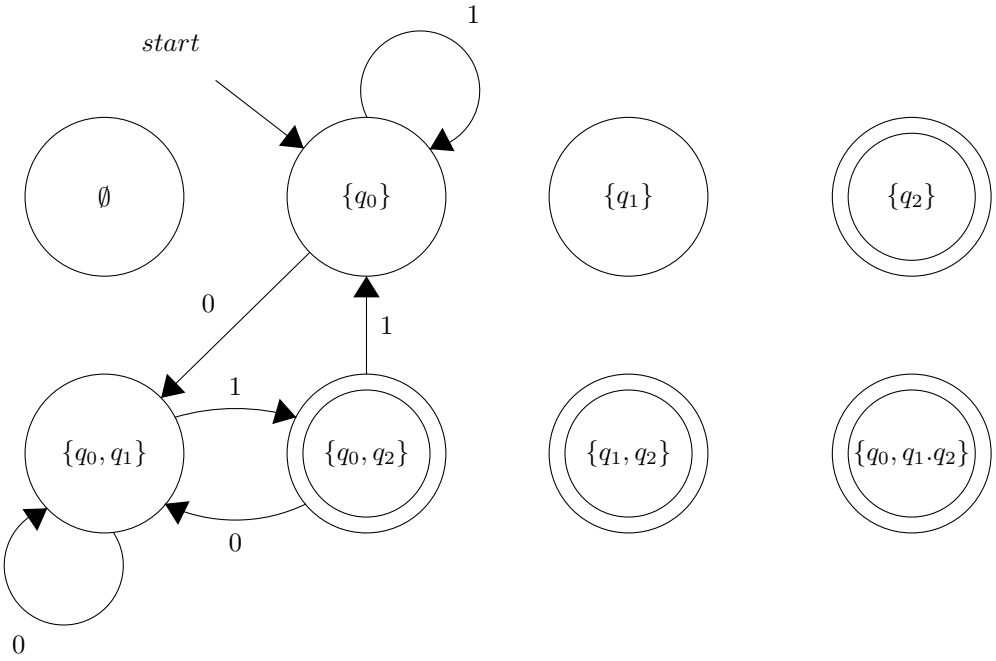
(b) What is the language recognized by this NFA?

Here, q_0 is an accepting state, so ϵ is accepted. However, since there are no outgoing transition from q_0, there's no accepting computations for any string of length > 0. The language recognized by this NFA is $\{\epsilon\}$.

Note: $\{\} \neq \{\epsilon\}$

Problem 3. Convert this NFA to an equivalent DFA using the subset construction:



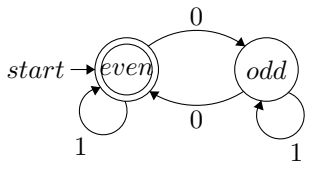


Don't forget to note the accept states!

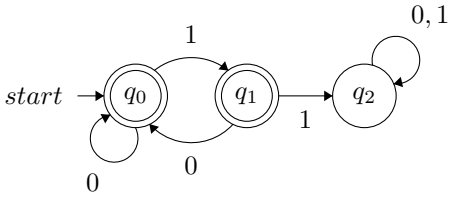
2 Closure Properties of regular languages

a) Closure under union

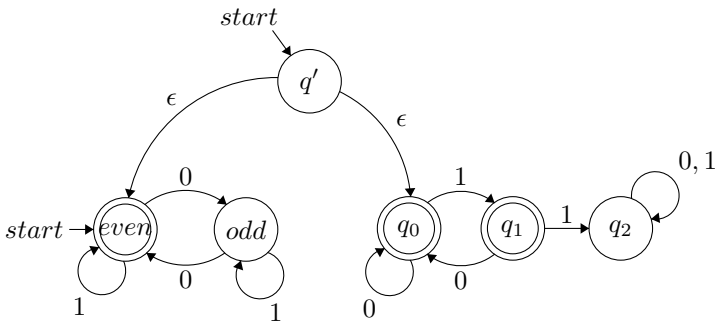
Problem 4.
$L_1 = \{w \in \{0,1\}^* | w$ has an even number of 0's$\}$
$L_2 = \{w \in \{0,1\}^* | w$ does not have two consecutive 1's$\}$



DFA for L_1



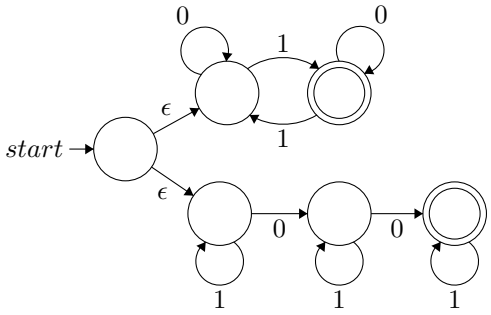
DFA for L_2



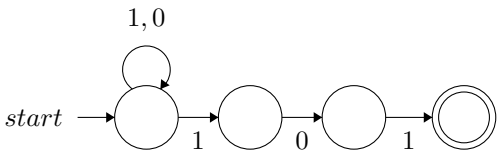
NFA for $L_1 \cup L_2$

b) Closure under concatenation

Problem 5.
$L_1 = \{w \in \{0,1\}^* | w$ has an odd number of $1's$ or exactly two 0s$\}$.
$L_2 = \{w \in \{0,1\}^* | w$ ends with 101$\}$.



NFA for L_1

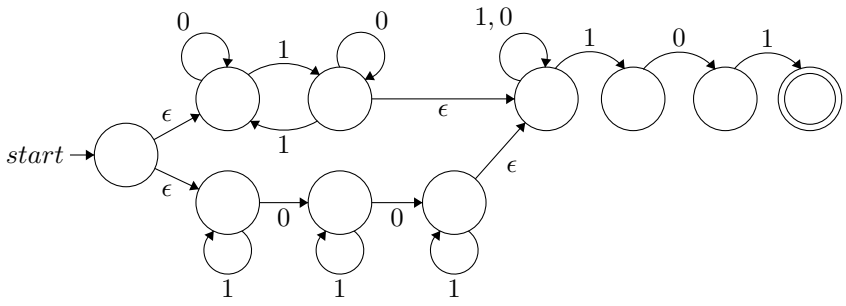


NFA for L_2

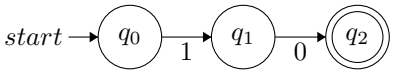
c) Closure under Kleene Star

Problem 6. We provide examples where the attempts fail – there are many other examples that work, of course.

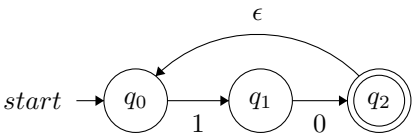
1. Let $L = \{10\}$. Then $L^* = \{(10)^n | n \geq 0\}$. Although $\epsilon \notin L$, by definition of Kleene star operation $\epsilon \in L^*$. Here, we see that attempt 1 does not accept the empty string.



NFA for $L_1 \cup L_2$

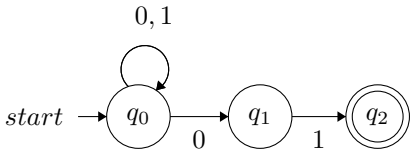


DFA for L

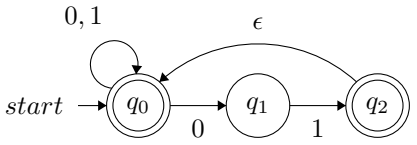


Incorrect attempt 1 for L^*

2. Let $L = \{w \in \{0,1\}^* |$ w ends with 01$\}$. Then $L^* = \{\epsilon\} \cup \{w \in \{0,1\}^* |$ w ends with 01$\}$. Let's draw the NFA for L and the wrong NFA for L^*.



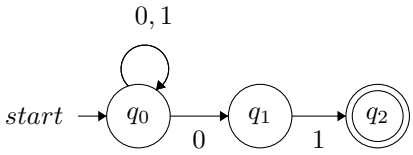
L



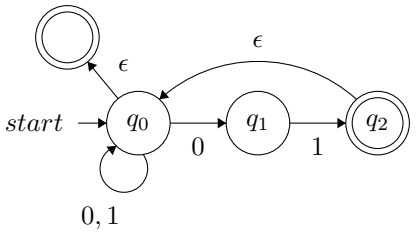
Incorrect attempt 2 for L^*

The NFA on the right accepts ϵ and the strings from $\{w \in \{0,1\}^* |$ w ends with 01$\}$. However, it accepts additional bad strings such as 000 which is not in L^*. In fact, it recognizes Σ^* because of q_0's transition to itself for 0 and 1. Hence the NFA on the right does not recognize L^*.

3. This construction is actually the same as the construction in attempt 2. For the same $L = \{w \in \{0,1\}^* |$ w ends with 01$\}$, let's draw the NFA for L and the wrong NFA for L^*.



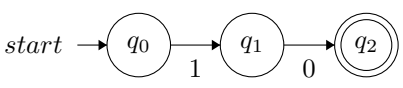
L



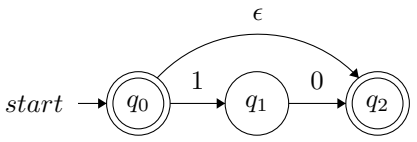
Incorrect attempt 3 for L^*

Again, while this incorrectly constructed NFA accepts ϵ and the strings from $\{w \in \{0,1\}^* |$ w ends with 01$\}$, it also accepts Σ^*. This is because any computation that reaches q_0 will automatically be accepted due to the ϵ-transition to the new accepting state.

4. Let's look at $L = \{10\}$. Then $L^* = \{(10)^n | n \geq 0\}$



L



Incorrect attempt 4 for L^*

The new NFA accepts strings $\{\epsilon, 10\}$. However, it does not accept any other string in L^* such as 1010.

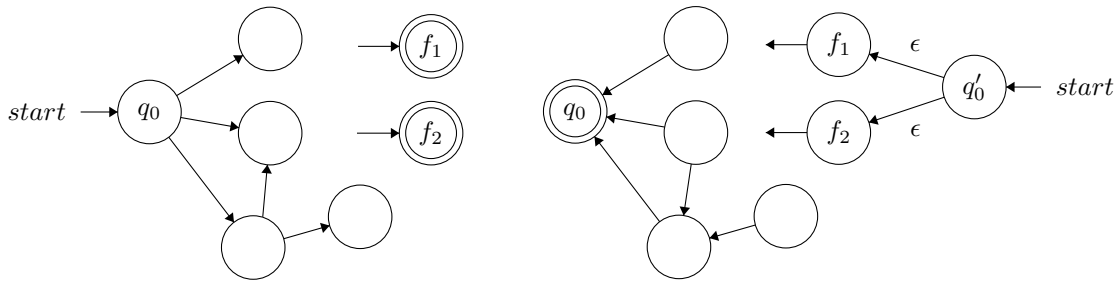
d) More operations

For 7a, I will give a very formal proof so that you have an idea on how to do these proofs in full detail.

Problem 7a (Reverse). Proof idea: to construct a DFA B that recognizes L^R, we duplicate the DFA A that recognizes L, reverse all the arrows, define start state of B as the accepting states of A, and define the accepting state of B as the start state of A. Since there could be multiple accepting states in A (which are now the "start states" in B and since there can only be one start state), we create a new start state that ϵ transitions to all these states, getting an NFA that recognizes L^R.

There exists a DFA $A = (Q_A, \Sigma, \delta_A, q_0, F_A)$ such that $L(A) = L$. Construct the following NFA $B = (Q_B, \Sigma, \delta_B, q_0', F_B)$ where:

- $Q_B = Q_A \cup \{q_0'\}$. We carry over the old states from A, but add a new start state q_0'.
- $\delta_B(q_0', \epsilon) = F_A$. The new start state has epsilon transitions to all the previously accepting states in A.
- For all $r \in Q_B$ and $a \in \Sigma$, $\delta_B(r, a) = \{q \in Q_A | \delta_A(q, a) = r\}$. We define the transition from each state $r \in Q_B$ as ending in all the states that have transition to the same r in A. In other words, we are reversing the arrows.
- $F_B = \{q_0\}$. The accepting state in B is the start state of A.



DFA for L NFA for L^R

Now we prove that NFA B recognizes L^R. To prove this we must show that:

$$w^R \in L^R \iff \text{NFA } B \text{ accepts } w^R$$

Forward Direction: $w^R \in L^R \implies$ NFA B accepts w^R.
Let $w = a_1, a_2 \ldots, a_n \in L$, $n \geq 0$ and $w^R = a^n, \ldots, a_2, a_1 \in L^R$. Since $w \in L$, for all $a_1, ..., a_n$ there exists some $q_i, r_i \in Q_A$ such that $\delta_A(q_i, a_{i+1}) = r_{i+1}$, $0 \leq i \leq n-1$ where q_0 is the start state of A and $r_n \in F_A$. And by our definition of NFA B, $r_n \in F_A$ implies that there exists a start state $q_0' \in Q_B$ such that $\delta(q_0, \epsilon) = r$. And again by our definition of B, for all $\delta_A(q_i, a_{i+1}) = r_{i+1}$, $0 \leq i \leq n-1$, there exists $\delta_B(r_{i+1}, a_{i+1}) = q_i \in Q_B$. And finally, since $F_B = \{q_0\}$, and there exists $\delta_B(r_1, a_1) = q_0$, the string $w^R = a_n, \ldots, a_2, a_1 \in L^R$ has an accepting computation in NFA B from its start state q_0' to its accepting state q_0.

Reverse Direction: NFA B accepts $w^R \implies w^R \in L^R$
This is similar to the forward direction but instead we argue that since NFA B accepts w^R, there is an accepting computation path from $q_0' \in Q_B$ to $F_B = \{\text{start state } q_0 \in Q_A\}$. (Being informal here) And by our construction of NFA B, then there must be a computation path in NFA A from $q_0 \in Q_A$ to $q_n \in F_A$ for the string $w = a_1 a_2 \ldots a_n$ and so $w \in L = L(A)$. If $w \in L$, then by definition of L^R, $w^R \in L^R$. We have prove that if B accepts w^R, then $w^R \in L^R$.

NFA B recognizes L^R and therefore L^R is regular.

Problem 7b. It is important to understand what $\min(L)$ does. When we say $w \in L$ and proper prefix of w is not in L, $\min(L)$ operation throws away all accepted strings whose prefixes can also be accepted. For example, if $q_i, q_j \in F_A$, and there is a string $w = xy \in L$ where $\delta_A^*(q_0, x) = q_i$, $\delta_A^*(q_i, x) = q_j$, then $w \notin \min(L)$. In other words, every transition from an accepting state will lead to a reject state.

For a DFA $A = (Q_A, \Sigma, \delta_A, q_0, F_A)$ such that $L(A) = L$, we will construct DFA $B = (Q_B, \Sigma, \delta_B, q_0', F_B)$ such that

- $Q_B = Q_A \cup \{q_{\text{reject}}\}$, where q_{reject} is the reject state. We need this additional reject state to capture all transitions coming from an accepting state.
- $q_0' = q_0$. The start state of A is the same as the start state of B
- $F_B = F_A$. A and B have the same accepting states.
- For all $a \in \Sigma$ and $q \in F_A = F_B$, $\delta_B(q, a) = q_{\text{reject}}$. All transitions coming from an accepting state will reach the reject state q_{reject}
- For all $a \in \Sigma$, $\delta_B(q_{\text{reject}}, a) = q_{\text{reject}}$. Once reached q_{reject}, the DFA's state does not change anymore.
- For all $a \in \Sigma$ and $q \notin F_B \cup q_{\text{reject}}$, $\delta_B(q, a) = \delta_A(q, a)$. Transitions from all other states remain unchanged.

To prove this is correct, we need to show that:

$$w \in \min(L) \implies \text{DFA } B \text{ accepts } w$$

$$w \notin \min(L) \implies \text{DFA } B \text{ rejects } w$$

First statement: $w \in \min(L) \implies$ DFA B accepts w.
Note that $\min(L) \subset L$. If $w \in min(L)$, then $w \in L$, and on input w DFA A ends in an accepting state $q \in F_A = F_B$. Secondly, since $w \in min(L)$, no proper prefix of w is in L. This means, for any x such that $w = xy, |y| > 0$, $x \notin L$. On input x DFA A will end on a state $q \notin F_A = F_B$. This means that when computing w, DFA A and B only reach accepting state on the last character. Therefore DFA B accepts w.

Second statement: $w \notin \min(L) \implies$ DFA B rejects w.
If $w \notin \min(L)$, then $w \notin L$ or there exist some proper prefix of $w \in L$.

- If $w \notin L$, then w will end on a non-accepting state in DFA A. Since $F_A = F_B$, this state is also non-accepting in DFA B. w will be rejected by DFA B.
- If there exists some proper prefix $w \in L$, then there is a smallest proper prefix $x \in L$ where $w = xy, |y| > 0$. Then DFA B will reject w because any the transitions after seeing x will go to and remain on q_{reject}.

3 Regular Expressions

Problem 8. Describe in words the language expressed by these regular expressions:
Answer.

(a) 0^*1^*
This is the set of binary strings that has any number of 0's, followed by any number of 1's. $\{0^n1^m | n, m \geq 0\}$

(b) $(01)^*$
This is the set of binary strings with repeating pattern of 01. $\{(01)^n | n \geq 0\}$

(c) $(0^*1^*)^*$
This is the set of all binary strings. $L = \{0, 1\}^*$

(d) $(0 \cup 1)^*$
This is also the set of all binary strings! Think about why the regular expression in part c represents the same language as part d

(e) $0^*(10^*1)^*0^*$
This is the set of all binary strings with even number of 1s

Problem 9. Construct regular expressions for the following languages.

(a) This is simple. We concatenate 10 with the language of all strings over the alphabet.
Answer: $10(0 \cup 1)^*$

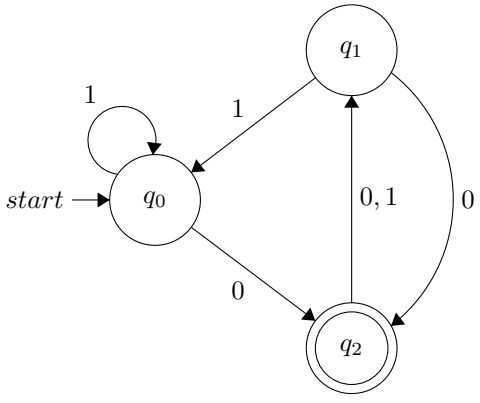
(b) Notice that the position of the two 0s can be anywhere in an accepted string. How do we express this in a regular expression? The idea is to fix the two 0's in a position and pad the 1s around it like this: $1^*01^*01^*$. This ensures that there can be any number of 1's in any position while there are only two 0's.
Answer: $1^*01^*01^*$

(c) Odd number of 0s mean there are $2n + 1$ number of 0's, where $n \geq 0$. So we can fix a single 0 in the expression and cover the remaining $2n$ using a Kleene star operation on two 0's. But it will not be sufficient to say something like $1^*01^*(00 \cup 1)^*$. This forces all the 0's except the first 0 to be in pairs (e.g. 1010011) and doesn't recoginze strings like 1010101. Instead, remember that we already showed how to recognize strings with only two zeroes (in any position) in the previous example. We can simply apply Kleene start to it to account for $2n$ number of 0s! Then add a 1^*0 to the front to account for the extra odd 0. **Exercise:** Does it matter if we add the odd 0 to the front or back? Why?
Answer: $1^*0(1^*01^*01^*)^*$

(d) This is a challenging one. Don't worry if you were not able to solve it on your own! The idea here is to observe the language and see what patterns can be exploited.

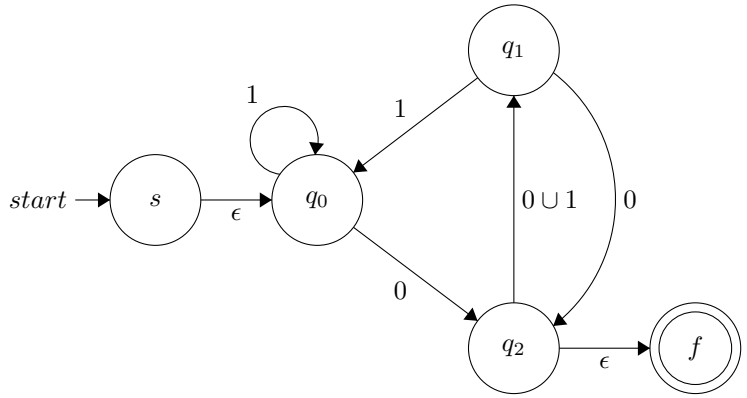
Let L be the language in question. The regular expression for the language with all strings over $\{a, b, c\}$ is $(a \cup b \cup c)^*$. I observe that for any occurance of b in L, I can prevent the formation of abc by adding c before b, a after b, or another b before and/or after b. This causes a substring like $\ldots abc \ldots$ to become $\ldots acbc \ldots$, $\ldots abac \ldots$, $\ldots abbc \ldots$ or $\ldots abbbc \ldots$. I can do this by $(a \cup cb \cup ba \cup bb \cup bbb \cup c)^*$. Note that $(bb \cup bbb)^*$ is the same as $\{b^n |$ where $n >= 2\}$. This accounts for legal substrings such as $ab^{2n+1}c$ where $n >= 1$. Now if we check back, there is no way for us to construct any string with substring abc. But we are not done yet. By forcing b's to be followed by b or a, or preceded by b or c, we have ommitted good strings that can start or end with b no matter what comes before it. Example, bc or ab. To fix this we add $(\epsilon \cup b)$ to the start and end to allow for the addition of b there.
Answer: $(\epsilon \cup b)(a \cup cb \cup ba \cup bb \cup bbb \cup c)^*(\epsilon \cup b)$

Problem 10. Convert the following NFA to a GNFA. Then convert the GNFA into a regular expression by ripping states (we note that ripping states in a different order can result in a different – but equivalent – regular expression).



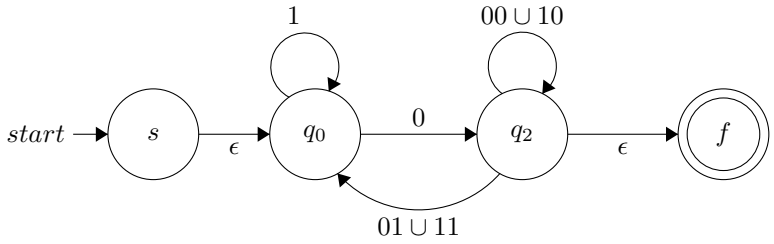
Step 1: Construct GNFA
Recall a GNFA is simply a NFA where the transition arrows may have any regular expressions as labels, instead of only $a \in \Sigma$ or ϵ. Given an arbitrary DFA or NFA, convert it to a GNFA as follows:

- New start state s with ϵ-transitions to original state
- New accest state f which will be the only accept state. Add ϵ-transitions from old accept states to f
- Replace transitions labeled with multiple symbols by transition labeled wih union of symbols
- All missing transitions labeled with $\emptyset$. We omit this here for clarity.



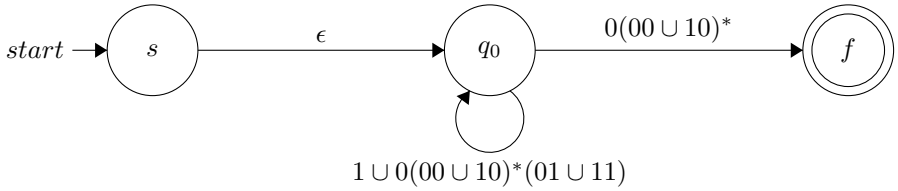
Step 2: Remove q_1
When we remove a state from a GNFA we have to account for all the transitions going into and coming out of that state. Hence, consider all pairs of edges $(q \to q_1, q_1 \to q')$ where $q, q' \neq q_1$

- $q_2 \to q_1$, $q_1 \to q_0$: $(0 \cup 1)1 \equiv 01 \cup 11$
- $q_2 \to q_1$, $q_1 \to q_2$: $(0 \cup 1)0 \equiv 00 \cup 10$



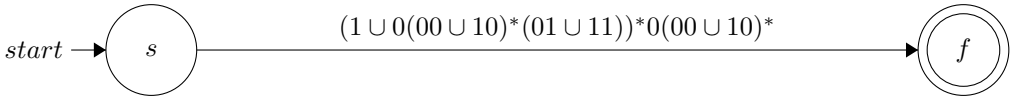
Step 3: Remove q_2

- $q_0 \to q_2$, $q_2 \to f$: $0(00 \cup 10)^*$
- $q_0 \to q_2$, $q_2 \to q_0$: $1 \cup 0(00 \cup 10)^*(01 \cup 11)$



Step 4: Remove q_4

- $s \to q_0$, $q_0 \to f$: $(1 \cup 0(00 \cup 10)^*(01 \cup 11))^*0(00 \cup 10)^*$

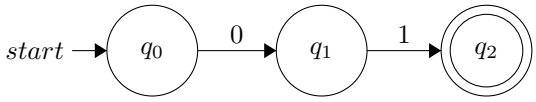


The regular expression from the start to finish state is $(1 \cup 0(00 \cup 10)^*(01 \cup 11))^*0(00 \cup 10)^*$

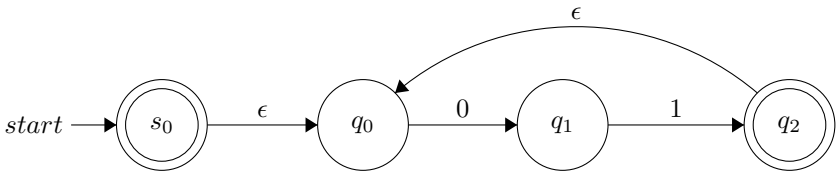
Problem 11. Convert $(01)^*(10^* \cup 0)$ into an NFA. We will not try to optimize the NFA or use our intuition about the language, but rather just use the constructions we saw in class for closure, in order to practice these constructions. The order of operation is $(),^*, \circ, \cup$. Hence, let's break down the regular expression appropriately.

1. Build NFA that recognizes $(01)^*$
   (a) First, construct NFA that recognizes 01
   (b) Apply Kleene star to get a NFA that recognizes $(01)^*$
2. Build NFA that recognizes $(10^* \cup 0)$
   (a) Build NFA that recognizes (10^*)
   (b) Build NFA that recognizes 0
   (c) Apply union operation to get NFA that recognizes $(10^* \cup 0)$
3. Apply concatenation to get NFA that recognizes $(01)^*(10^* \cup 0)$

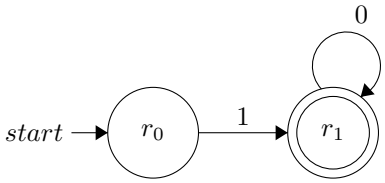
Step 1a: Construct NFA that recongizes 01



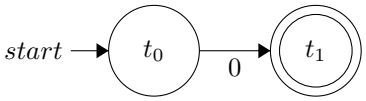
Step 1b: Apply Kleene star to get a NFA that recognizes $(01)^*$
To apply Kleene star operation, we add a new start state that ϵ-transitions to the old start state and the accepting states ϵ-transitions back to the start state s_0. The new start state is also an accepting state.



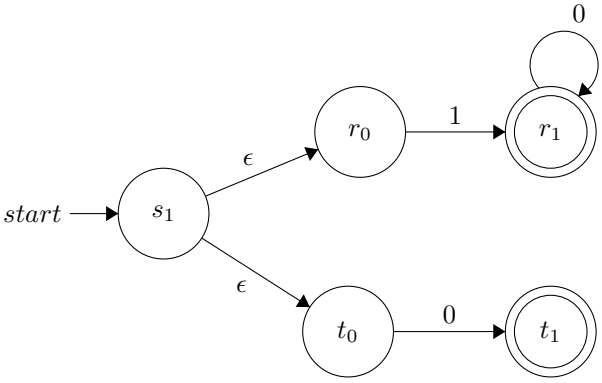
Step 2a: Build NFA that recognizes (10^*)



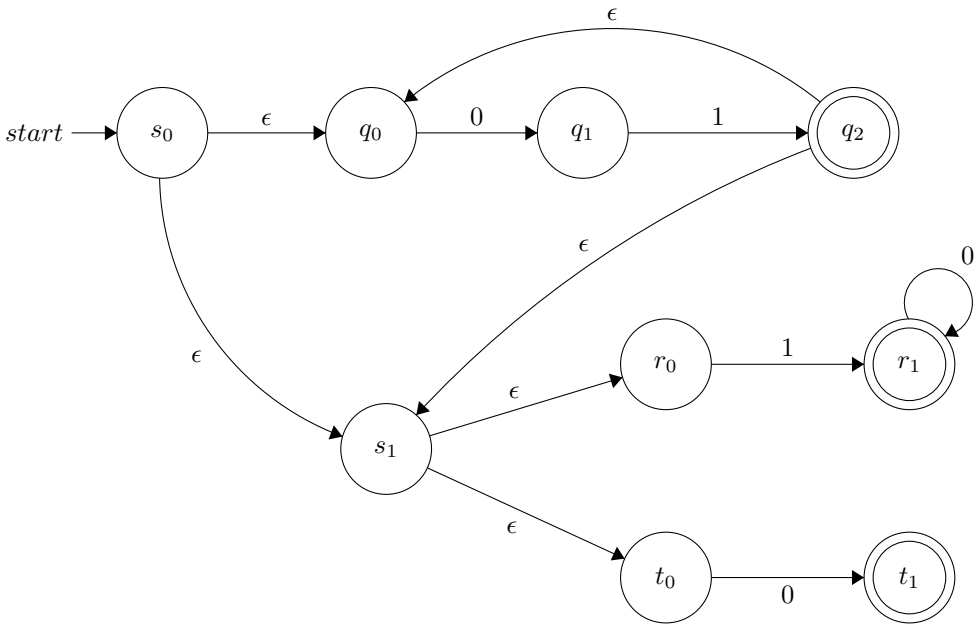
Step 2b: Build NFA that recognizes 0



Step 2c: Apply union operation to get NFA that recognizes $(10^* \cup 0)$
To apply union, we add an extra start state that epsilon transitions to all the old start states.



Step 3: Apply concatenation to get NFA that recognizes $(01)^*(10^* \cup 0)$
To apply concatenation, add ϵ-transition from accepting states s and q_2 to the state s_1. Then remove s_0 and q_2 from the set of accepting states for the new NFA



The above NFA recognizes the language generated by $(01)^*(10^* \cup 0)$