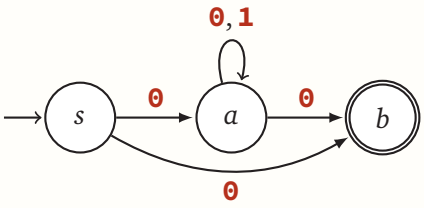
1. Let $L = \{w \in \{\mathtt{0},\mathtt{1}\}^* \mid w \text{ starts and ends with } \mathtt{0}\}$.

   (a) Construct an NFA for L with exactly three states.

Solution: The following NFA N accepts the language. On seeing the symbol **0**, the NFA has moves to a. The NFA stays at a when reading **1**; on reading **0** it has the choice to stay or decide that this is the last character and move to b.



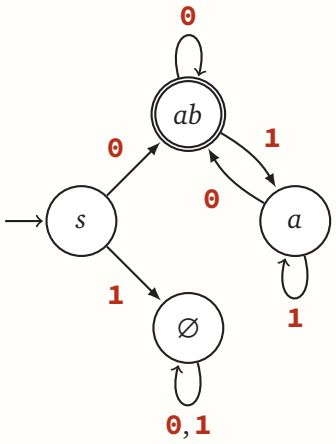
■

   (b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have four states, all reachable from the start state.

Solution:

q'	ε-reach	**0**	**1**	A'?
s	s	ab	$\varnothing$	
ab	ab	ab	a	✓
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	
a	a	ab	a	

Thus we obtain the following four-state DFA; each DFA-state is labeled with the corresponding set of NFA-states:

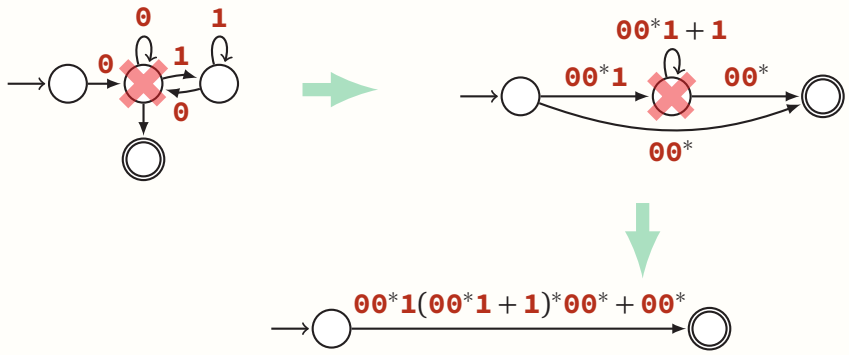


■

(c) Convert the DFA you constructed in part (b) into a regular expression using the state elimination algorithm.

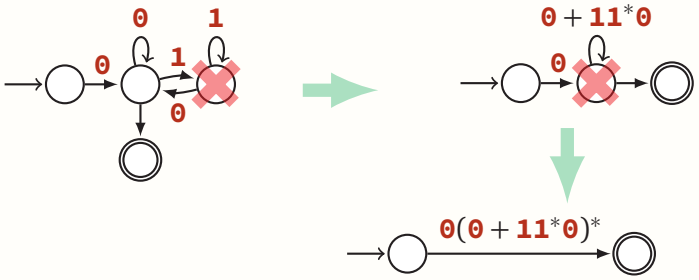
Solution: Unlabeled arrows indicate ε-transitions.

The starting state has no incoming transitions, so we just need to add an isolated ending state. After removing the dump state $\emptyset$ we eliminate ab then a.



We end with the regular expression $\mathtt{00}^*\mathtt{1}(\mathtt{00}^*\mathtt{1}+\mathtt{1})^*\mathtt{00}^*+\mathtt{00}^*$.

Eliminating a then ab gives a different (but equivalent) regular expression:



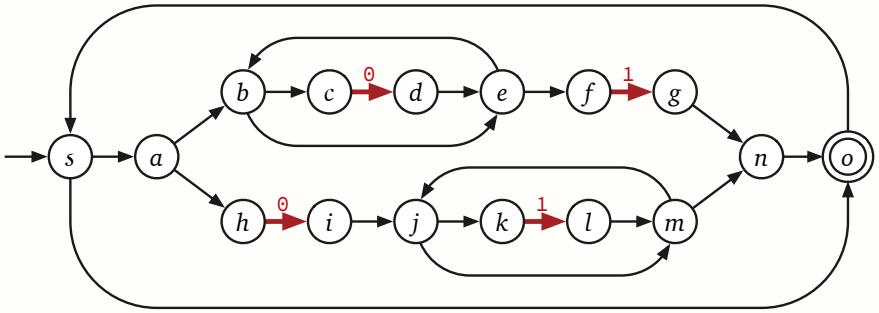
We end with the equivalent regular expression $\mathtt{0}(\mathtt{0}+\mathtt{11}^*\mathtt{0})^*$. ■

(d) Write a simpler regular expression for L.

Solution: Applying the identity $A+BB^*A = B^*A$ to the second regular expression above gives $\mathtt{0}(\mathtt{1}^*\mathtt{0})^*$. ■

2. (a) Convert the regular expression $(\mathtt{0}^*\mathtt{1} + \mathtt{01}^*)^*$ into an NFA using Thompson's algorithm.

Solution: Here unlabeled arrows indicate ε-transitions:



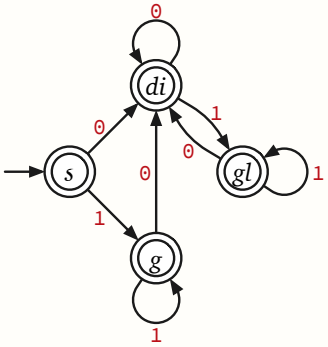
■

(b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have four states, all reachable from the start state. (Some of these states are obviously equivalent, but keep them separate.)

Solution:

q'	ε-reach	$\mathtt{0}$	$\mathtt{1}$	A'?
s	$sabcefho$	di	g	✓
di	$sabcdefhijkmno$	di	gl	✓
g	$sabcefghno$	di	g	✓
gl	$sabcefghjklmhno$	di	gl	✓

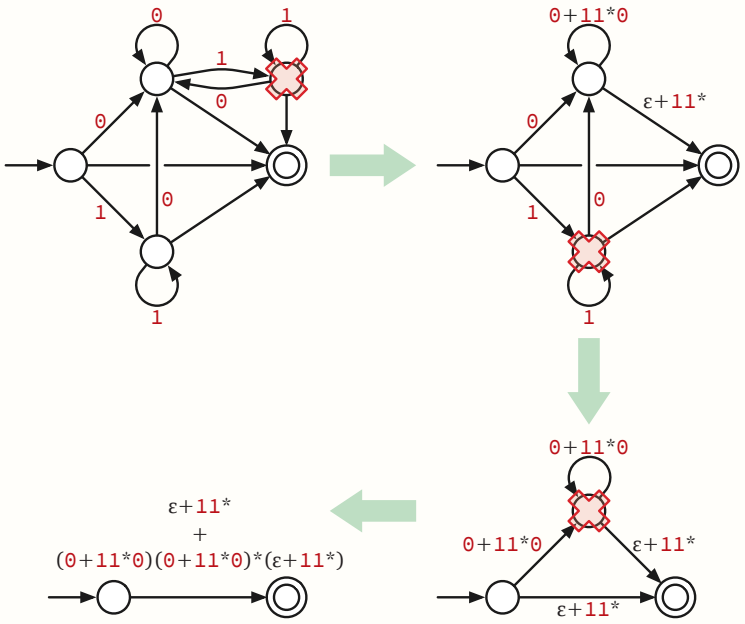
We obtain the following 4-state DFA; here each DFA-state is labeled with the corresponding set of NFA-states:



Obviously all four states are equivalent, because they're all accepting states, but we're not supposed to collapse them. ■

(c) Convert the DFA you just constructed into a regular expression using the state elimination algorithm. You should *not* get the same regular expression you started with.

Solution: As usual, unlabeled arrows indicate ε-transitions. We start by adding a unique accepting state (with ε-transitions from the old accepting states), and then eliminate one state at a time.



We end with the regular expression $\varepsilon + \mathbf{11}^* + (\mathbf{0} + \mathbf{11}^*\mathbf{0})(\mathbf{0} + \mathbf{11}^*\mathbf{0})^*(\varepsilon + \mathbf{11}^*)$.

Applying the equivalence $A + BB^*A = B^*A$ three times simplifies this regular expression to $(\mathbf{1}^*\mathbf{0})^*\mathbf{1}^*$. ■

(d) **Think about later:** Find the smallest DFA that is equivalent to your DFA from part (b) and convert that smaller DFA into a regular expression using the state elimination algorithm. Again, you should *not* get the same regular expression you started with.

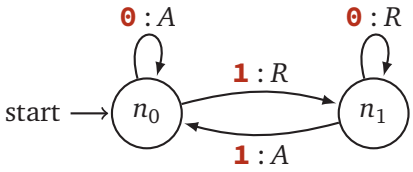
Solution: Because every state in the DFA from part (b) is accepting, the minimal equivalent DFA has only one state.

Running the state elimination algorithm on this trivial DFA yields the regular expression $(\mathbf{0} + \mathbf{1})^*$. ■

(e) What is this language?

Solution: All binary strings. ■

3. In a previous lab/homework we talked about a new machine called a *finite-state transducer* (FST). The special part thing about this type of machine is that it gives an output on the transition instead of the state that it is in. An example of a finite state transducer is as follows:



defined by the five tuple: $(\Sigma, \Gamma, Q, \delta, s)$. Let's constrain this machine (call is FST_{AR}) a bit and say the output alphabet consists of two signals: accept or reject ($\Gamma = \{A, R\}$). We say that $L(FST_{AR})$ represents the language consisting of all strings that end with a accept (A) output signal.

Prove that $L(FST_{AR})$ represents the class of regular languages.

Solution: The lovely thing about this problem is that it is harder the more you think about it and the key is simply to address it in small chunks. So let's break it down into parts. First let's define some notations about the machines we have available:

- DFA: $M = (Q_M, \Sigma_M, \delta_M, s_M, A_M)$
- NFA: $N = (Q_N, \Sigma_N, \delta_N, s_N, A_N)$
- FST_{AR} $F = (Q_F, \Sigma_F, \Gamma_F, \delta_F, s_F)$. For the transistion function let's say $\delta(q, a) = (q, b)$ where $a \in \Sigma$ and $b \in \Gamma$

OK so to prove a machine represents regular languages, we must show two things:

(a) A FST_{AR} can represent any regular language.

(b) Any language that is accepted by a FST_AR is regular.

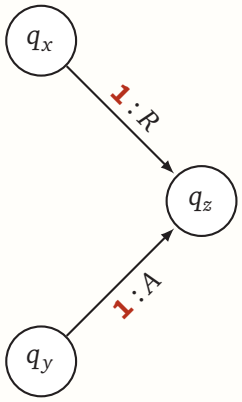
Part (a): A FST_{AR} can represent any regular language: Multiple ways to do this but the easiest is by using a language transformation like you did in lab. If we can show a construction that turns any DFA into a FST_{AR}, then we show that a FST_{AR} can represent any regular language.

Doing this is relatively straightforward. Accept in a DFA is ending in an accept state. Accept in a FST_{AR} is ending on an accept transition.

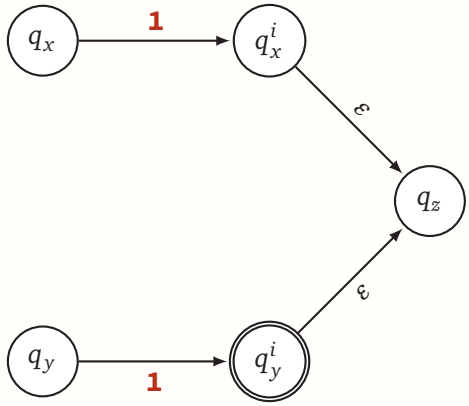
Hence, given a DFA (M), we'd like to construct a FST_AR (F) such that $L(M) = L(F)$. We define F as follows:

$$
\begin{aligned}
Q_F &= Q_M \\
\Sigma_F &= \Sigma_M \\
s_F &= s_M \\
\Gamma &= \{A, R\} \\
\delta_F(q, a) &= (\delta_M(q, a), A) && \text{if } \delta_M(q, a) \in A_M \\
&= (\delta_M(q, a), R) && \text{if } \delta_M(q, a) \notin A_M
\end{aligned}
$$

Part (b): Any language that is accepted by a FST_AR is regular. Little harder this one. The immediate impulse is to turn the FST_{AR} into a NFA directly which is a good impulse. But because you can have multiple transitions going to the same state with mixed accept/reject signals, it'll be tough to know what states to make an accept state and what to make a reject state:



So a bit of a problem since we can't just turn q_z into an accepting state. But what if we add intermediate states on the path of the transitions like so:



This NFA-part denotes the same thing as the FST-part above it. So let's define the NFA formally:

$$Q_N = Q_F \times \Sigma \cup Q_F$$ Original states plus state per transition
Added states marked as (q_F, a)
Original states marked as $(q_F, \square)$

$$\Sigma_N = \Sigma_F$$
$$s_N = s_F$$
$$\delta_N((q, \square), a) = (q, a)$$ Original transitions go to intermediate states
$$\delta_N((q, a), \varepsilon) = (\delta_F(q, a)[0], \square)$$ Intermediate states go to expected original states
$$A_N = (q, a) \text{ if } \delta_F(q, a)[1] = A$$
Add intermediate states that correspond to accepting transition

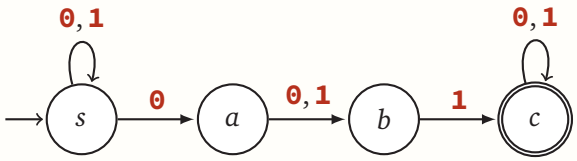
This construction shows that every language accepted by a FST_{AR} can also be accepted by an NFA.

Summation: Parts (a) and (b) together show that FST_{AR}'s represent the class of regular languages. ■

4. Let $L = \{w \in \{\mathtt{0},\mathtt{1}\}^* \mid$ a $\mathtt{0}$ appears in some position i of w, and a $\mathtt{1}$ appears in position $i+2\}$.

(a) Construct an NFA for L with exactly four states.

Solution: The following NFA N accepts the language. On seeing the symbol $\mathtt{0}$, the NFA has the choice of either staying at s or to check if it is followed, 2 positions later, with a $\mathtt{1}$.



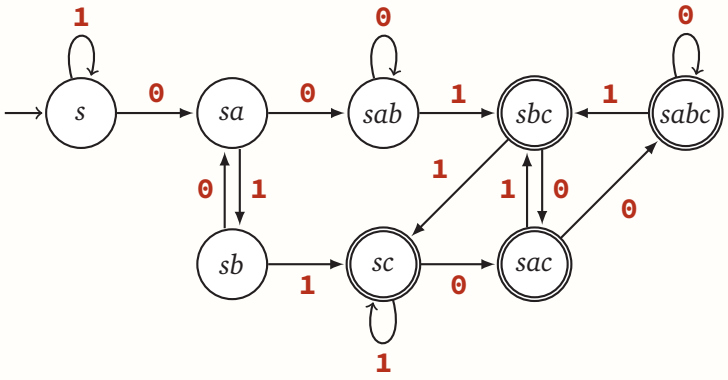
■

(b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have eight states, all reachable from the start state.

Solution:

q'	ε-reach	$\mathtt{0}$	$\mathtt{1}$	A'?
s	s	sa	s	
sa	sa	sab	sb	
sab	sab	sab	sbc	
sb	sb	sa	sc	
sbc	sbc	sac	sc	✓
sc	sc	sac	sc	✓
sac	sac	$sabc$	sbc	✓
$sabc$	$sabc$	$sabc$	sbc	✓

Thus we obtain the following eight-state DFA; each DFA-state is labeled with the corresponding set of NFA-states:

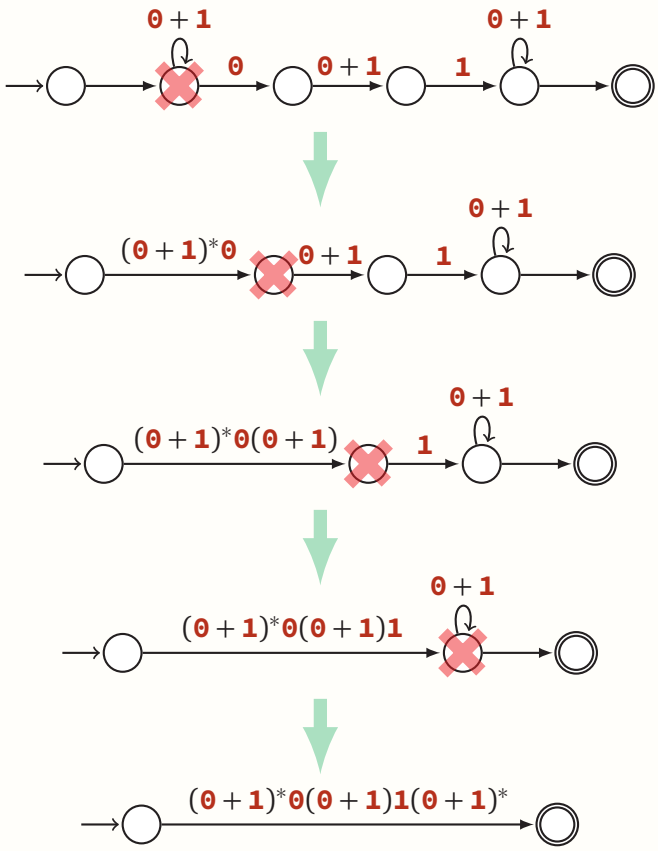


■

(c) Convert the NFA you constructed in part (a) into a regular expression using the state elimination algorithm.

Solution: As usual, unlabeled arrows indicate ε-transitions.

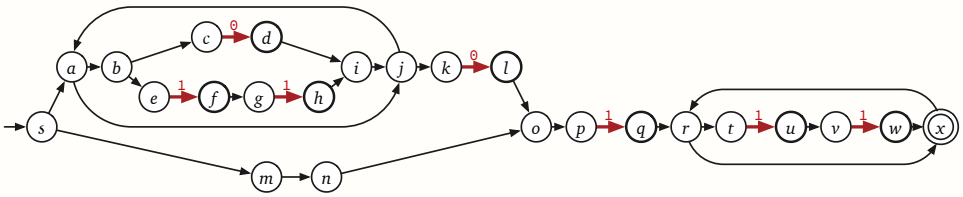
We begin by adding new isolated start and accepting states, and then eliminate left to right.



We end with the regular expression $(0+1)^*0(0+1)1(0+1)^*$. ■

5. (a) Convert the regular expression $(\varepsilon + (\mathtt{0} + \mathtt{11})^*\mathtt{0})\mathtt{1}(\mathtt{11})^*$ into an NFA using Thompson's algorithm.

Solution: Again, unlabeled arrows indicate ε-transitions:



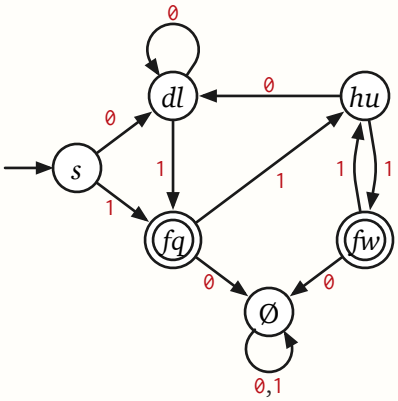
■

(b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have six states, all reachable from the start state. (Some of these states are obviously equivalent, but keep them separate.)

Solution:

q'	ε-reach	$\mathtt{0}$	$\mathtt{1}$	A'?
s	$sabcejkmnop$	dl	fq	
dl	$abcedijklop$	dl	fq	
fq	$fgqrtx$	$\emptyset$	hu	✓
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	
hu	$abcehijkuv$	dl	fw	
fw	$fgrtwx$	$\emptyset$	hu	✓

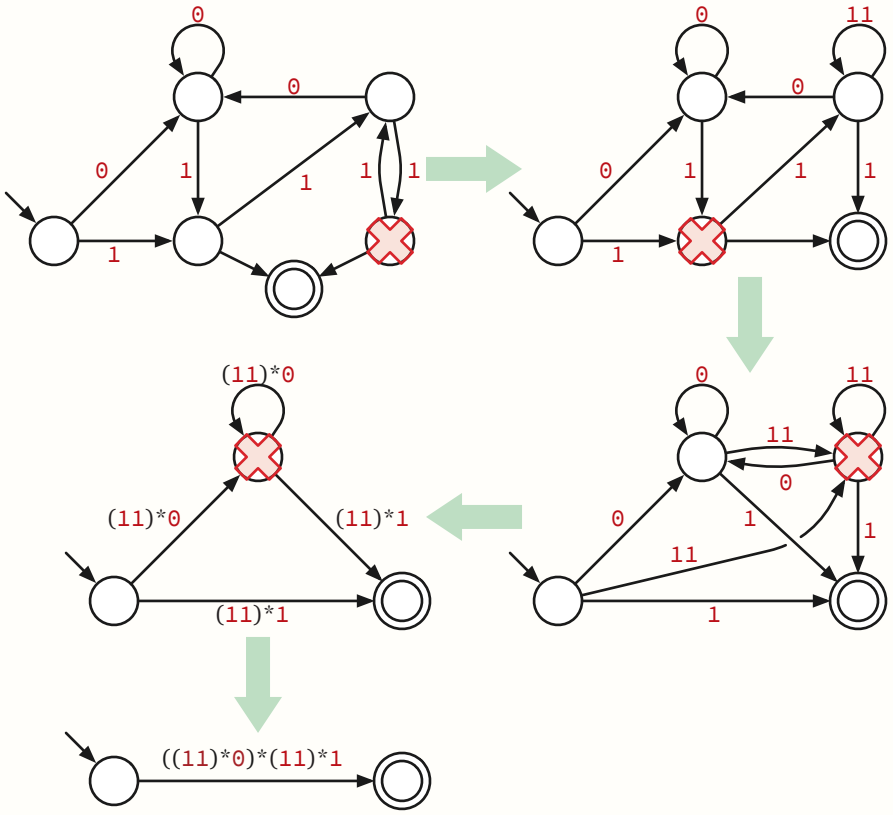
Thus, we obtain the following six-state DFA; each DFA-state is labeled with the corresponding set of NFA-states:



■

(c) Convert the DFA you just constructed into a regular expression using the state elimination algorithm. You should *not* get the same regular expression you started with.

Solution: As usual, unlabeled arrows indicate ε-transitions. After removing the dump state $\emptyset$ and adding a unique accepting state (with ε-transitions from the old accepting states), we eliminate one state at a time. To simplify the final expression, I applied the equivalence $A + BB^*A = B^*A$ on the fly in the last two stages of the algorithm.



We end with the regular expression $((\mathbf{11})^*\mathbf{0})^*(\mathbf{11})^*\mathbf{1}$.

(Eliminating the states in different orders yields different regular expressions.) ■

(d) What is this language?

Solution: All binary strings that end with an odd-length run of **1**s, where all other runs of **1**s have even length. More simply: $(\mathbf{0} + \mathbf{11})^*\mathbf{1}$. ■