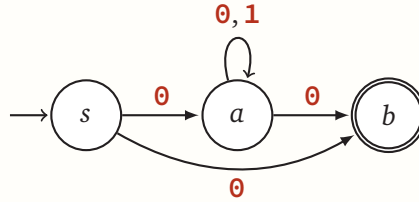


1. Let $L = \{w \in \{0, 1\}^* \mid w \text{ starts and ends with } 0\}$.

(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 .

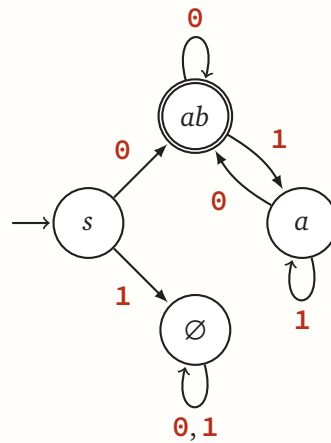


(b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have four states, all reachable from the start state.

Solution:

q'	ϵ -reach	0	1	A' ?
s	s	ab	\emptyset	
ab	ab	ab	a	✓
\emptyset	\emptyset	\emptyset	\emptyset	
a	a	ab	a	

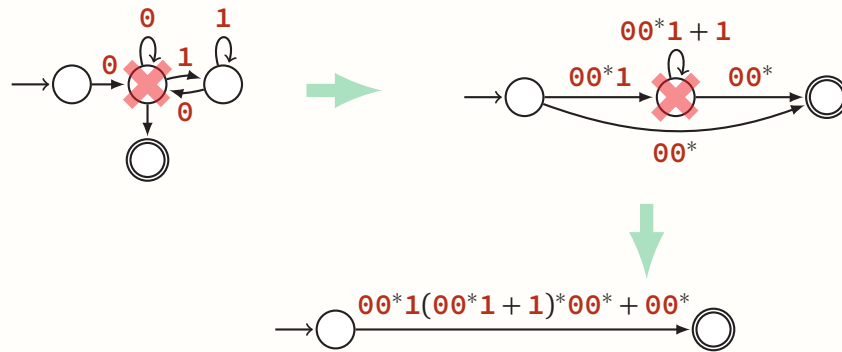
Thus we obtain the following four-state DFA; each DFA-state is labeled with the corresponding set of NFA-states:



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

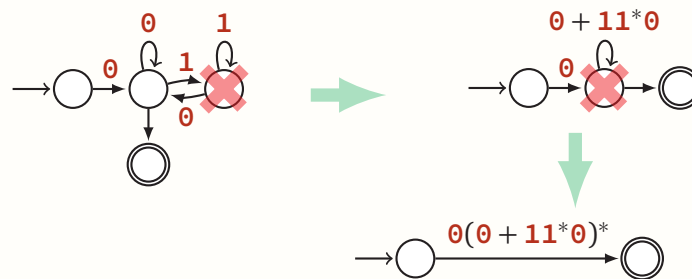
Solution: Unlabeled arrows indicate ϵ -transitions.

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 $00^*1(00^*1+1)^*00^*+00^*$.

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



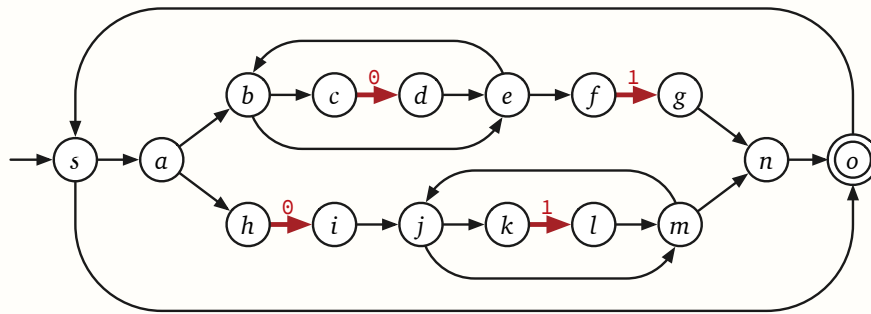
We end with the equivalent regular expression $0(0+11^*0)^*$. ■

- (d) Write a simpler regular expression for L .

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

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

Solution: Here unlabeled arrows indicate ϵ -transitions:

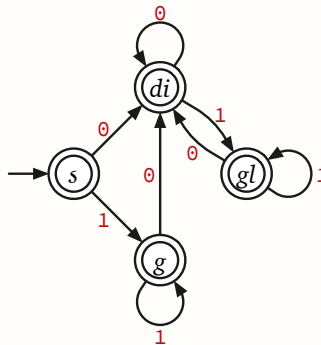


- (b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have four states, all reachable from the start state. (Some of these states are obviously equivalent, but keep them separate.)

Solution:

q'	ϵ -reach	0	1	A' ?
s	$sabcefho$	di	g	✓
di	$sabcdehijklmno$	di	gl	✓
g	$sabcefghno$	di	g	✓
gl	$sabcefgijklmno$	di	gl	✓

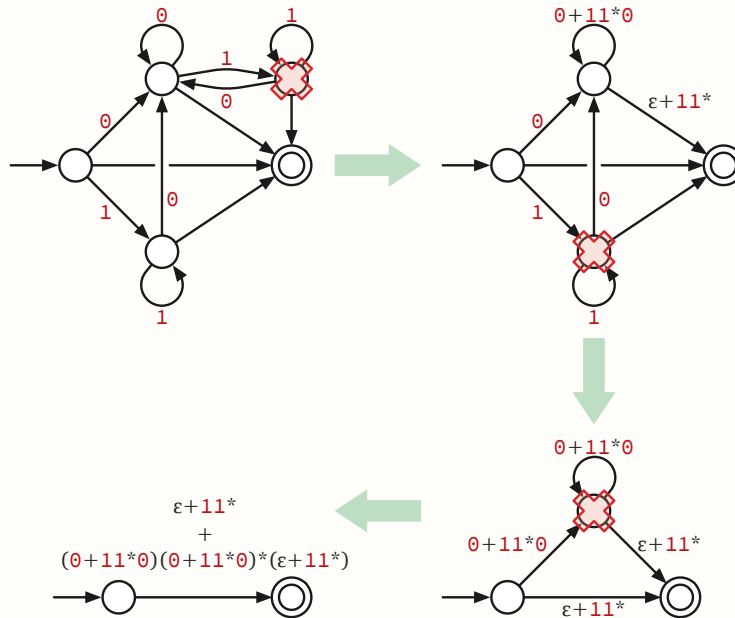
We obtain the following 4-state DFA; here each DFA-state is labeled with the corresponding set of NFA-states:



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 $\epsilon + 11^* + (0 + 11^*0)(0 + 11^*0)^*(\epsilon + 11^*)$. Applying the equivalence $A + BB^*A = B^*A$ three times simplifies this regular expression to $(1^*0)^*1^*$. ■

- (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.



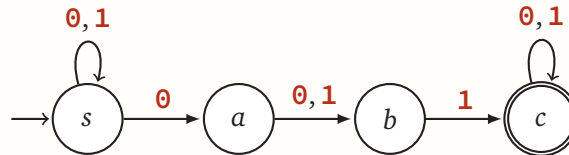
Running the state elimination algorithm on this trivial DFA yields the regular expression $(0 + 1)^*$. ■

- (e) What is this language?

Solution: All binary strings. ■

3. Let $L = \{w \in \{0, 1\}^* \mid \text{a } 0 \text{ appears in some position } i \text{ of } w, \text{ and a } 1 \text{ appears in position } i + 2\}$.
- (a) Construct an NFA for L with exactly four states.

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

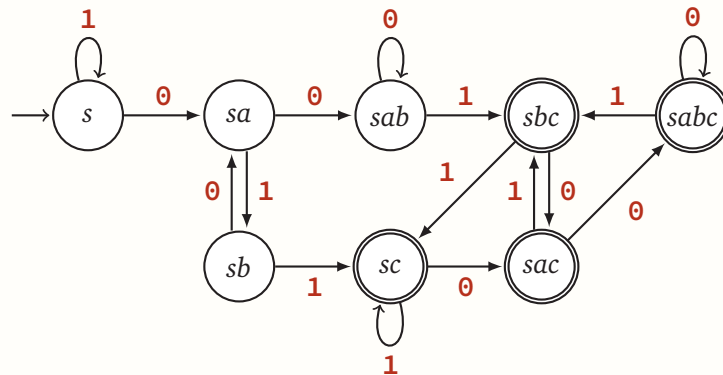


- (b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have eight states, all reachable from the start state.

Solution:

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

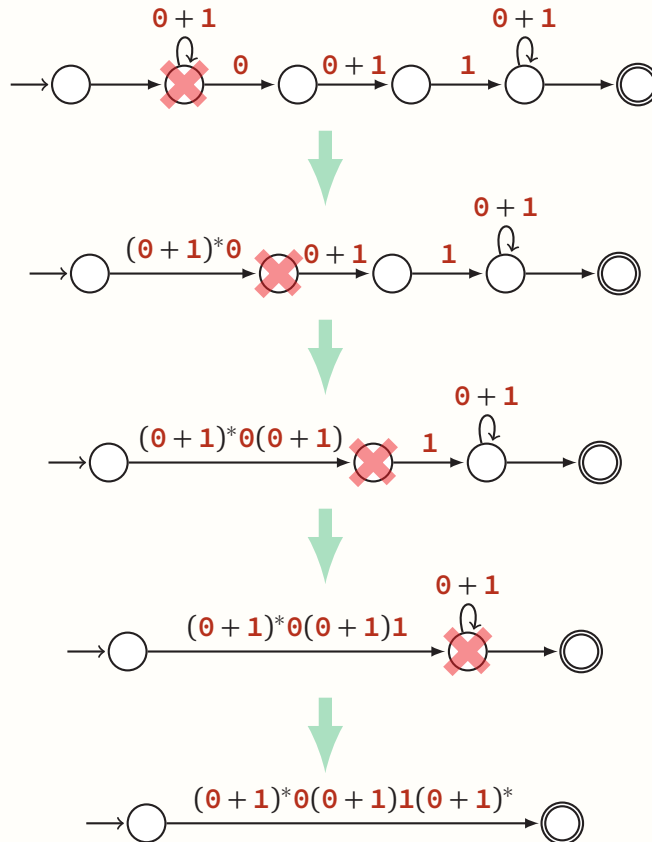
Thus we obtain the following eight-state DFA; each DFA-state is labeled with the corresponding set of NFA-states:



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

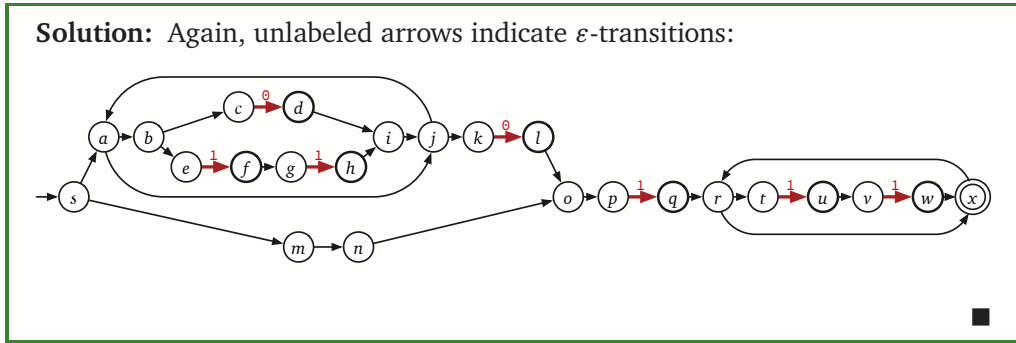
Solution: As usual, unlabeled arrows indicate ε -transitions.

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

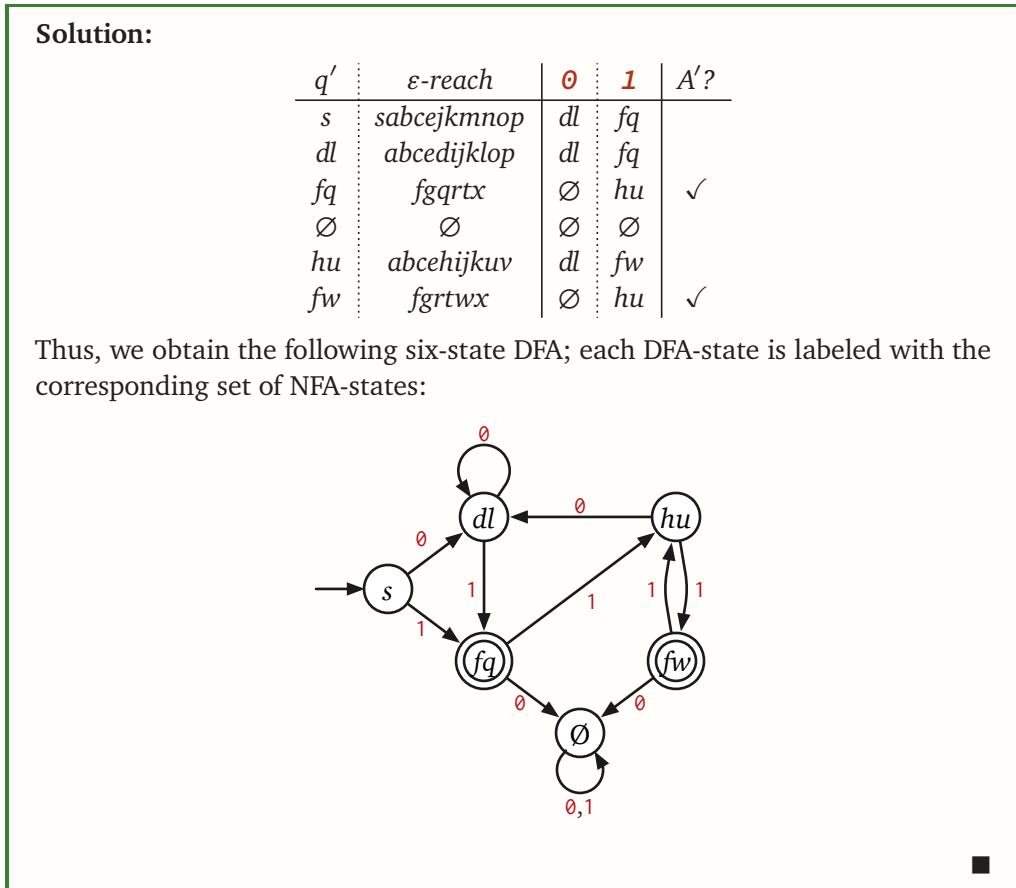


We end with the regular expression $(0+1)^*0(0+1)1(0+1)^*$. ■

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

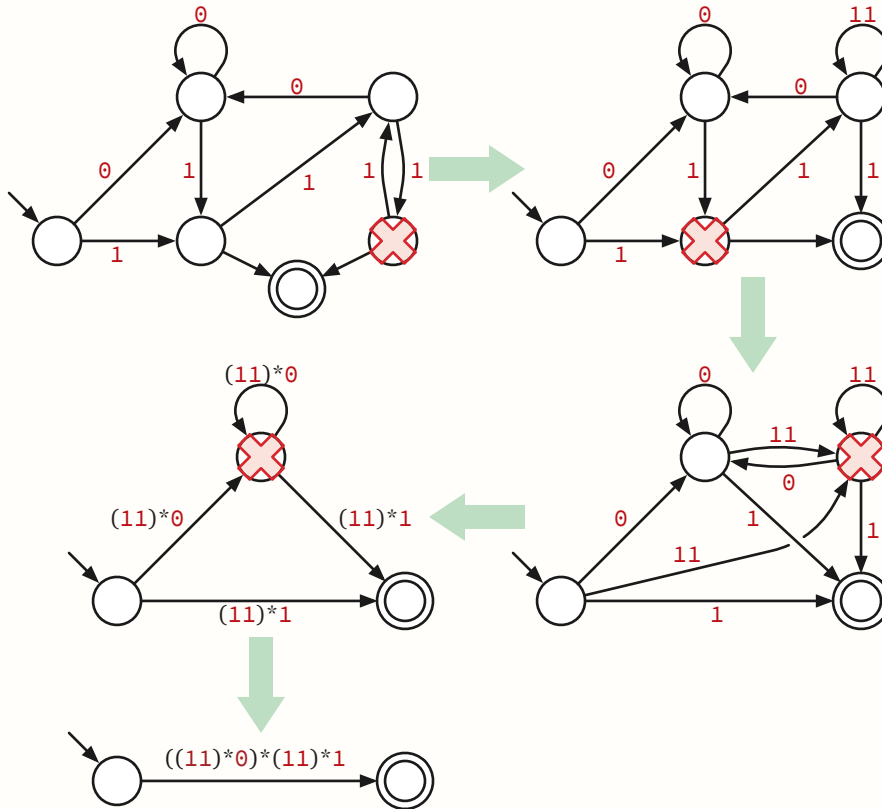


- (b) Convert the NFA you just constructed into a DFA using the incremental subset construction. Draw the resulting DFA. Your DFA should have six states, all reachable from the start state. (Some of these states are obviously equivalent, but keep them separate.)



- (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 $((11)^*0)^*(11)^*1$.

(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: $(0 + 11)^*1$. ■