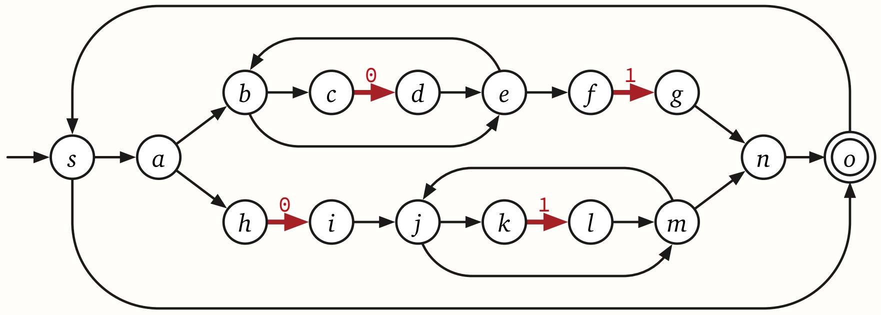
1. (a) Convert the regular expression $(0^*1 + 01^*)^*$ into an NFA using Thompson's algorithm.

Solution: Here unlabeled arrows indicate ε-transitions:



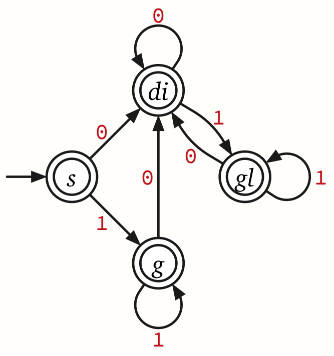
■

(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	$sabcdefhijkmno$	di	gl	✓
g	$sabcefghno$	di	g	✓
gl	$sabcefghjklmhno$	di	gl	✓

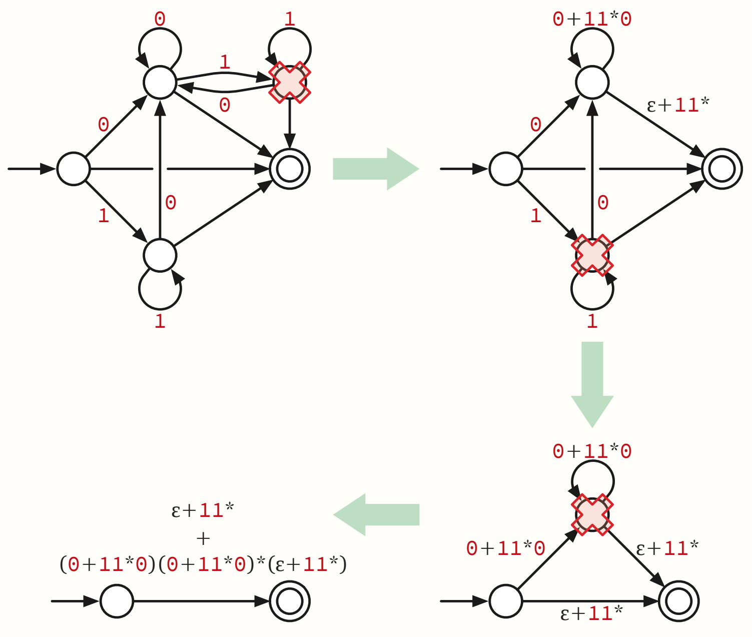
We obtain the following 4-state DFA; here each DFA-state is labeled with the corresponding set of NFA-states:



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

(c) **Think about later:** Convert the DFA you just constructed into a regular expression using Han and Wood's 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.

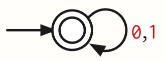


We end with the regular expression $\varepsilon + 11^* + (0 + 11^*0)(0 + 11^*0)^*(\varepsilon + 11^*)$.

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 Han and Wood's 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.



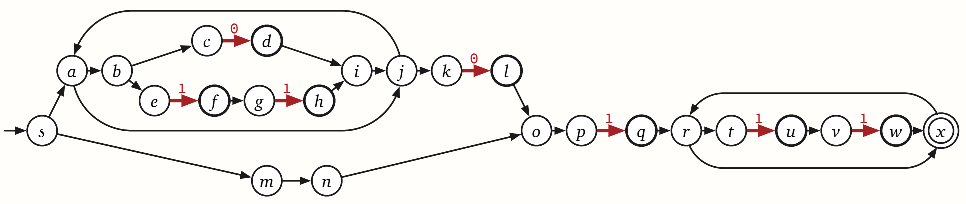
Running Han and Wood's algorithm on this trivial DFA yields the regular expression $(0 + 1)^*$. ■

(e) What is this language?

Solution: All binary strings. ■

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

Solution: Again, unlabeled arrows indicate ε-transitions:



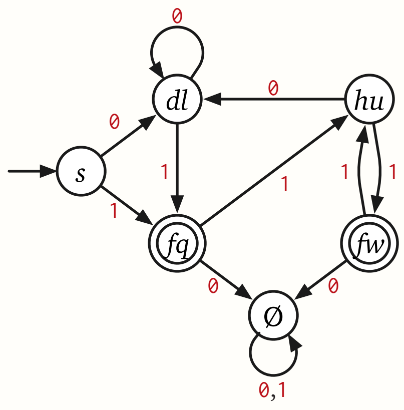
■

(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	0	1	A'?
s	$sabcejkmnop$	dl	fq	
dl	$abcedijklop$	dl	fq	
fq	$fgqrtx$	$\varnothing$	hu	✓
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	
hu	$abcehijkuv$	dl	fw	
fw	$fgrtwx$	$\varnothing$	hu	✓

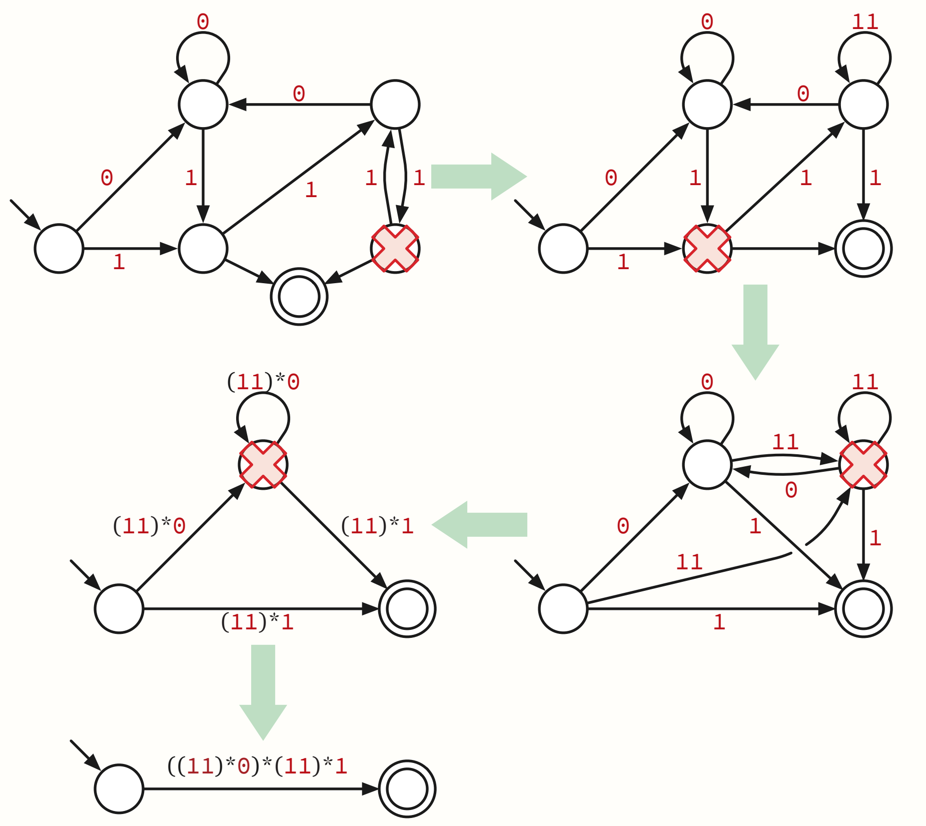
Thus, we obtain the following six-state DFA; each DFA-state is labeled with the corresponding set of NFA-states:



■

(c) **Think about later:** Convert the DFA you just constructed into a regular expression using Han and Wood's 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 $\varnothing$ 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) **Think about later:** Find the smallest DFA that is equivalent to your DFA from part (b), using Moore's algorithm (in Section 3.10 of the notes), and convert that smaller DFA into a regular expression using Han and Wood's algorithm. Again, you should *not* get the same regular expression you started with.

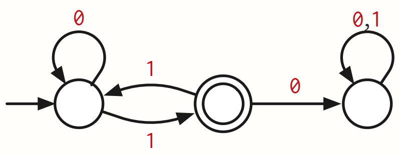
Solution: Initially, Moore's algorithm distinguish between accepting and non-accepting states. The first iteration of the main algorithm distinguishes non-accepting states by whether $\delta(q, 1)$ is accepting or not. Further iterations of the algorithm make no more progress, so we're done.

	s	*dl*	*hu*	*fq*	*fw*
dl					
hu					
fq	×	×	×		
fw	×	×	×		
∅				×	×

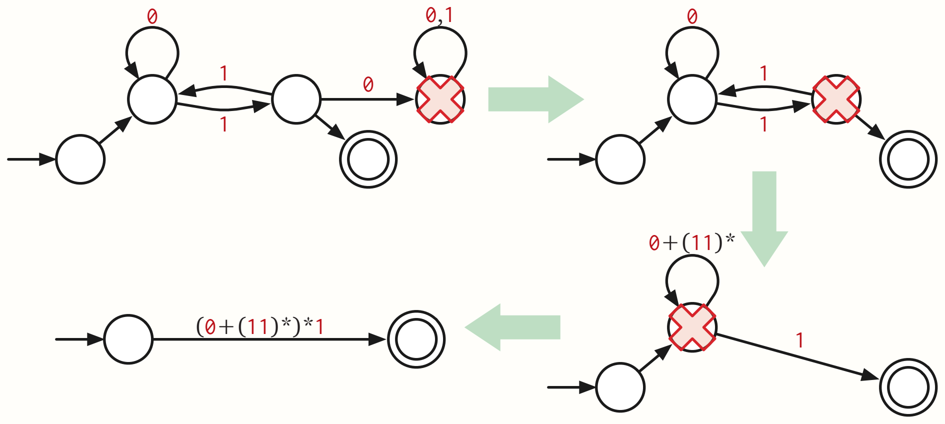
$\Longrightarrow$

	s	*dl*	*hu*	*fq*	*fw*
dl					
hu					
fq	×	×	×		
fw	×	×	×		
∅	×	×	×	×	×

The algorithm partitions the states into three equivalence classes: non-accepting states $\{s, dl, hu\}$, the accepting states $\{fq, fw\}$, and the trash state $\varnothing$.



Before we start Han and Wood's algorithm, we add new isolated start and accepting states. We then eliminate the old dump state, the old accepting state, and the old start state in that order



We end with the regular expression $(0 + (11)^*)^*1$.

If instead we eliminate the original start state before the original accepting state, we end with the regular expression $0^*1(10^*1)^*$. ■

(e) What is this language?

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