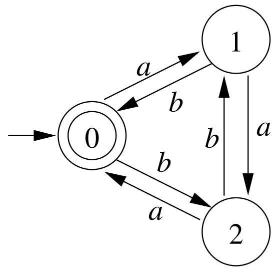
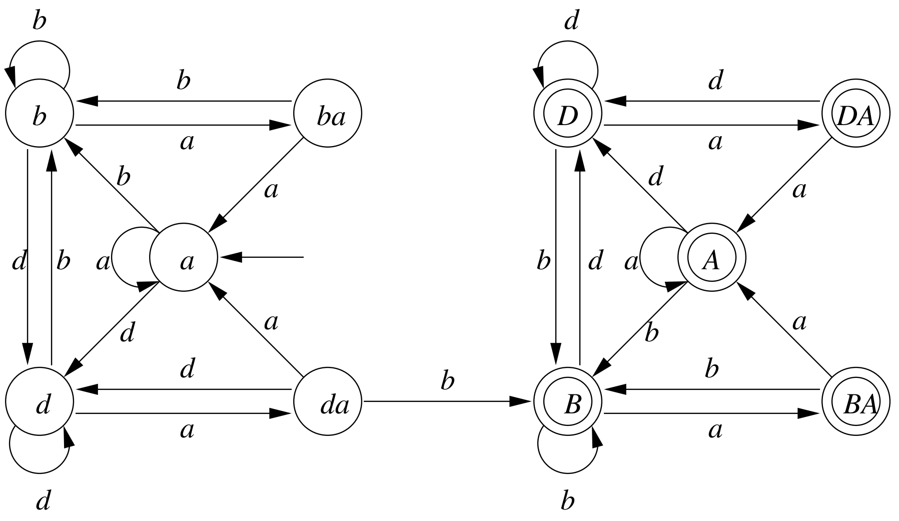
1. (a) $L = \{s \in \{a,b\}^* :$ the number of a's in s is equal to the number of b's in s, modulo $3\}$

   (b) 

   (c) The second DFA keeps track of the number of a's *minus* the number of b's, modulo 3, making use of the facts that $-1 \bmod 3 = 2$ and $-2 \bmod 3 = 1$.

2. (a) **NFA**: 

   In the block of states on the left, the states are used to keep track of the last one or two characters. More precisely:

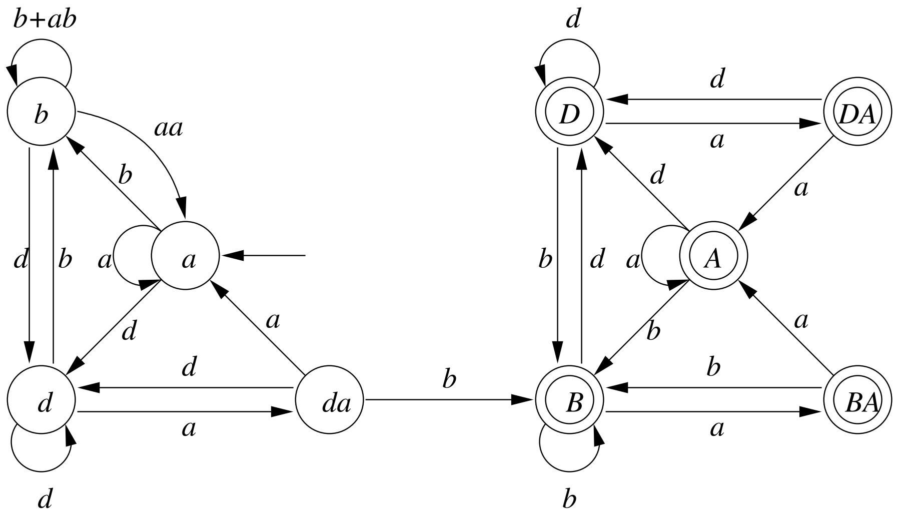
   - state "a" is reached if the string is empty, is equal to a, or ends with aa;
   - state "b" is reached if the string ends with b (but **not** dab);
   - state "d" is reached if the string ends with d (but **not** bad);
   - state "ba" is reached if the string ends with ba;
   - state "da" is reached if the string ends with da.

   The block of states on the right is reached if, and only if, the string contains the substring dab. Otherwise, the states are used in the same way as in the block of states on the left.

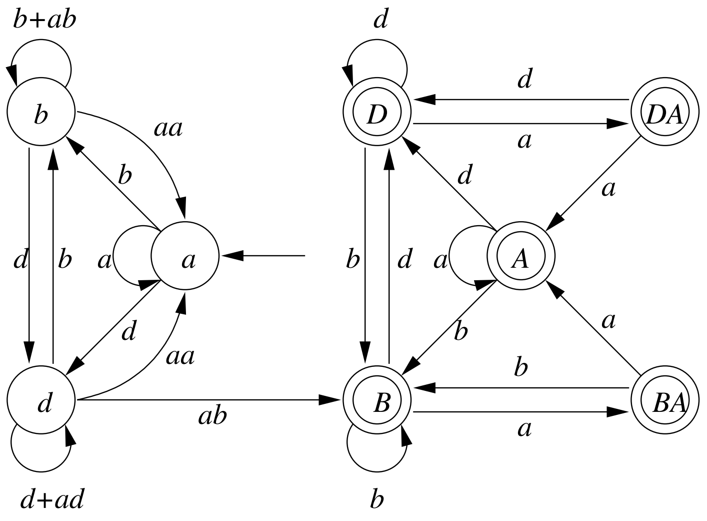
   Note that there is an implicit transition to a dead state from state "ba" when reading a d, from state "BA" when reading a d, and from state "DA" when reading a b (as this would represent a second occurrence of dab).

   RE: obtained from the NFA by using the construction described in class.

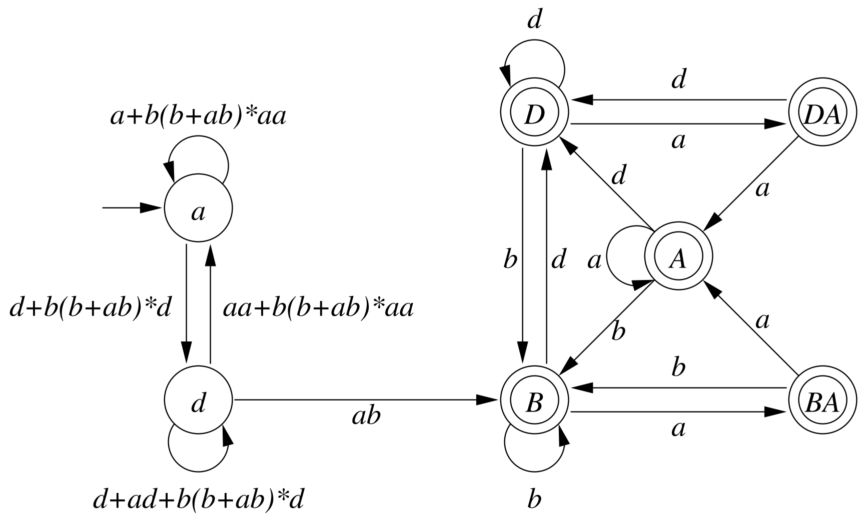
First, eliminate state "*ba*":



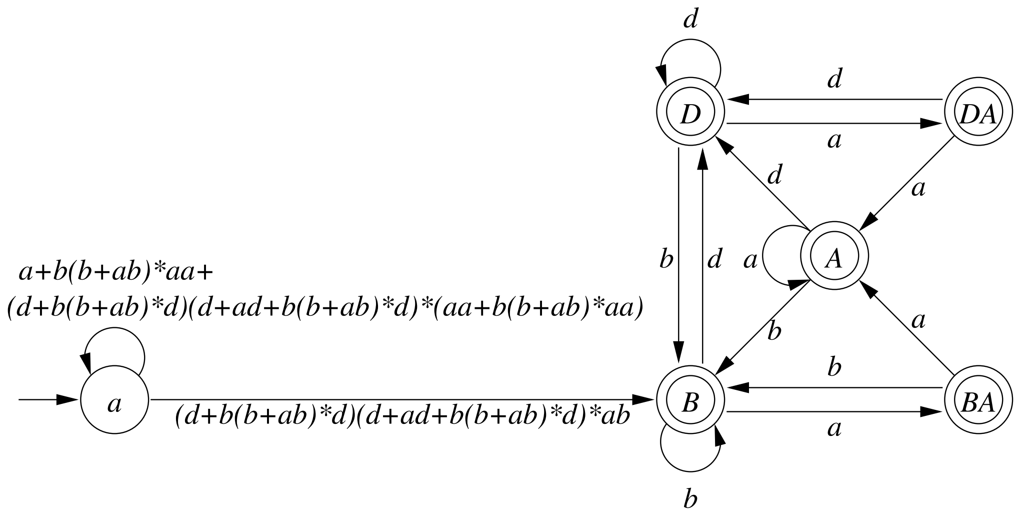
Next, eliminate state "*da*":



Next, eliminate state "*b*":



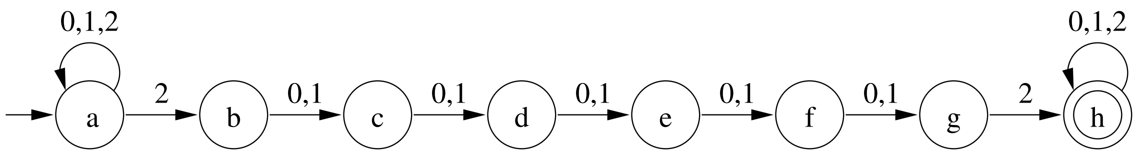
Next, eliminate state "*d*":


a+b(b+ab)*aa+
(d+b(b+ab)*d)(d+ad+b(b+ab)*d)*(aa+b(b+ab)*aa)
(d+b(b+ab)*d)(d+ad+b(b+ab)*d)*ab
a
B
D
A
DA
BA
a
b
d

Before we proceed further, it would be good to simplify some of the REs we've built up so far. (This is not strictly necessary, but it will make it easier to continue.)

. . . TO BE COMPLETED. . . This will be straightforward (it does not require any creativity) but tedious (it will require great attention to details and many steps, given the size of the NFA and the number of accepting states).

(b) **NFA**:

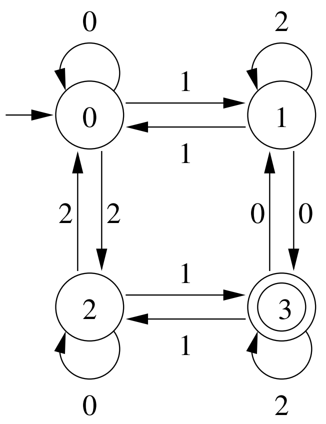


The only way for this NFA to accept is to process, at some point, a 2 followed by five characters other than a 2, followed by another 2. And the NFA does accept every string that has this property.

RE: $(0+1+2)^*2(0+1)(0+1)(0+1)(0+1)(0+1)2(0+1+2)^*$

The RE describes all the strings that contain, at some point, a 2 followed by five characters other than a 2, followed by another 2 — and no other string.

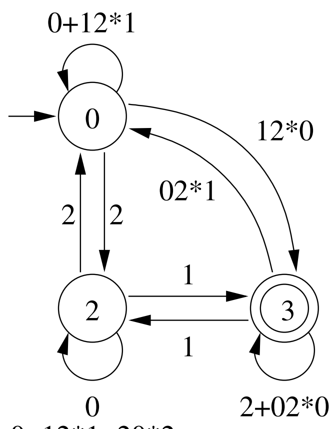
(c) **NFA**:



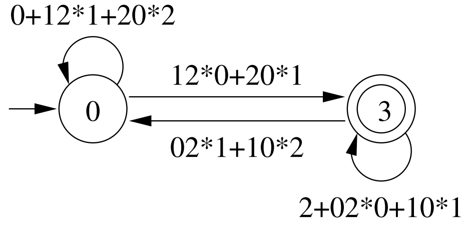
The states of the NFA keep track of the value of s in ternary notation, modulo 4. The transitions are based on the fact that, for all strings $s \in \{0,1,2\}^*$ and all characters $i \in \{0,1,2\}$, $(si) = 3(s) + i$ (where "(s)" denotes the ternary value of s).

RE: obtained from the NFA by using the construction described in class.

First eliminate state 1:



Next eliminate state 2:



The final RE is (using some square brackets for readability):

$$(0 + 12^*1 + 20^*2)^*(12^*0 + 20^*1)\big[2 + 02^*0 + 10^*1 + (02^*1 + 20^*2)(0 + 12^*1 + 20^*2)^*(12^*0 + 20^*1)\big]^*$$

3. We prove the statement "$\exists R_I, L(R_I) = \text{INIT}(L(R))$" by structural induction on the regular expression R.

Base Cases:

- $\text{INIT}(L(\varnothing)) = \varnothing = L(\varnothing)$.
- $\text{INIT}(L(\varepsilon)) = \{\varepsilon\} = L(\varepsilon)$.
- $\text{INIT}(L(a)) = \{\varepsilon, a\} = L(\varepsilon + a)$ for all $a \in \Sigma$.

Ind. Hyp.: Assume S and T are arbitrary regular expressions and S_I and T_I are regular expressions that satisfy $L(S_I) = \text{INIT}(L(S))$, $L(T_I) = \text{INIT}(L(T))$.

Ind. Step:

- If $R = S + T$, then $\text{INIT}(L(R)) = \text{INIT}(L(S) \cup L(T)) = \text{INIT}(L(S)) \cup \text{INIT}(L(T))$ because $\exists y, xy \in L(S) \cup L(T) \Leftrightarrow (\exists y, xy \in L(S)) \vee (\exists y, xy \in L(T))$.
  Hence, $L(S_I + T_I) = \text{INIT}(L(S + T))$ so we can let $R_I = S_I + T_I$.
- If $R = ST$, then $\text{INIT}(L(R)) = \{x \in \Sigma^* : \exists y \in \Sigma^*, \exists z_1 \in L(S), \exists z_2 \in L(T), xy = z_1 z_2\}$. But then, either x is some initial portion of z_1 (in which case $x \in L(S_I)$) or x consists of z_1 followed by some initial portion of z_2 (in which case, $x \in L(ST_I)$).
  Hence, $L(S_I + ST_I) = \text{INIT}(L(ST))$ so we can let $R_I = S_I + ST_I$.
- If $R = S^*$, then every string in $\text{INIT}(L(S^*))$ can be completed to become a string made up of some number of strings from $L(S)$ concatenated to each other. Every such string must consist of some number of strings from $L(S)$ followed by some initial portion of a string from $L(S)$.
  Hence, $L(S^* S_I) = \text{INIT}(L(S))$ so we can let $R_I = S^* S_I$.

By structural induction, for all regular expressions R, $\exists R_I, L(R_I) = \text{INIT}(L(R))$, *i.e.*, regular languages are closed under the INIT operation.