Design NDFA's which accept:

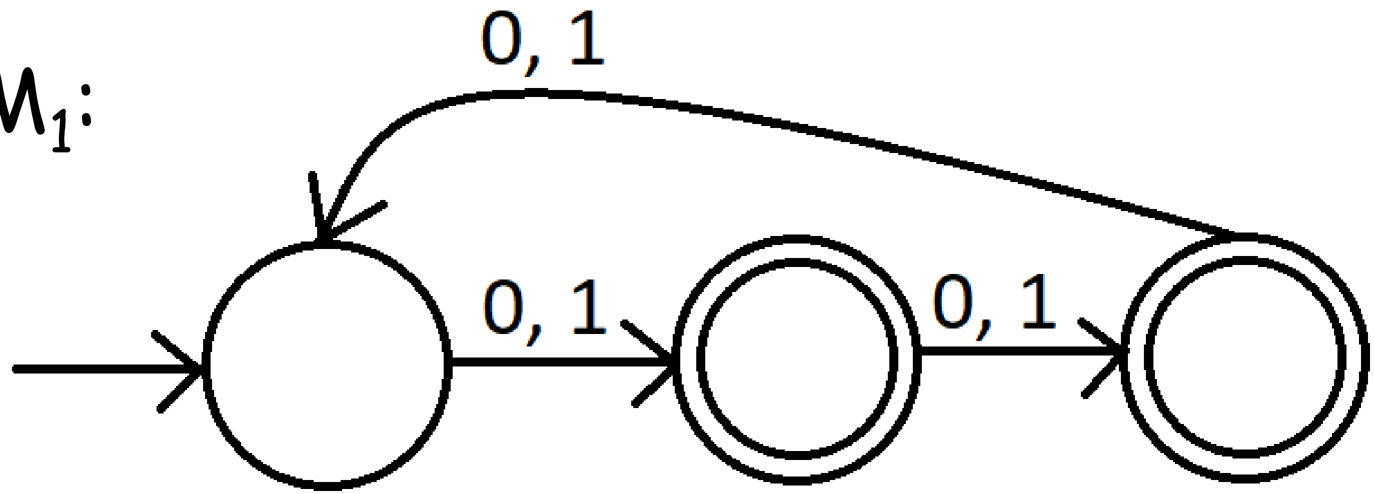
$L_1 = \{ w \in \{a, b\}^* :$ w contains one or more of aabb, abab, or abba as substrings$\}$

$L_2 = \{ w \in \{0, 1\}^* :$ w= u v for some u such that the length of u is even and for some v which contains 3k+1 0's$\}$

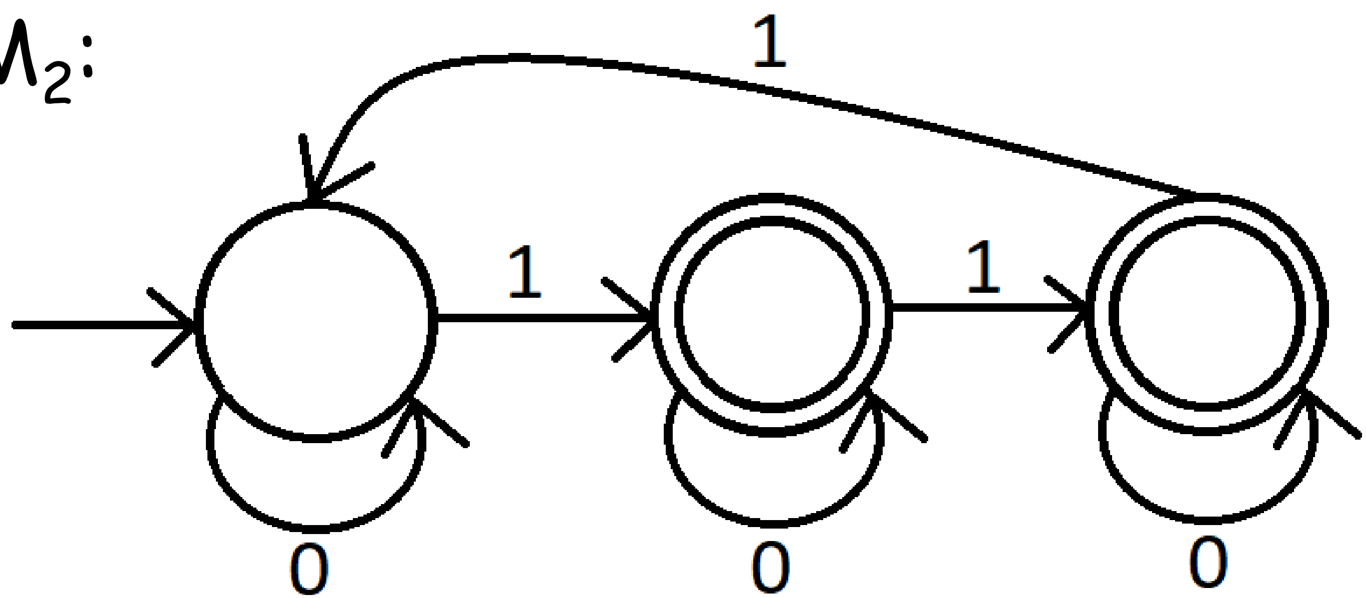
$L_3 = a\ (ab)^*\ a \quad \cup \quad ab\ (a \cup b\ )^*\ a$

# What languages do these DFA's accept?

$M_1$:




$M_2$:



Assignment #2 has been posted.
Due Friday June 2 at the beginning of class.

Tutorial #3 has been posted.

The next tutorial is Tuesday May 30.

For practice with regular expressions or DFA's, try the java-based tutorial available from the course web page.

# Regular Expression: L = { w an element of {a,b}* : w has both baa and aaba as a substring }.

L = { w an element of {a,b}* : w has both baa and aaba as a substring }.

Give a regular expression for L.

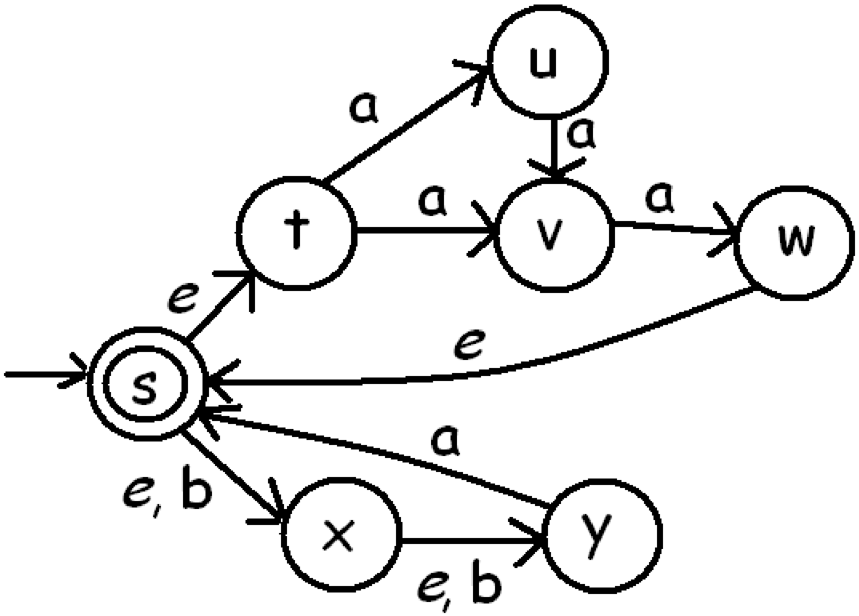
Your Answer: (a|b)*baa(a|b)*aaba|(a|b)*aaba(a|b)*baa

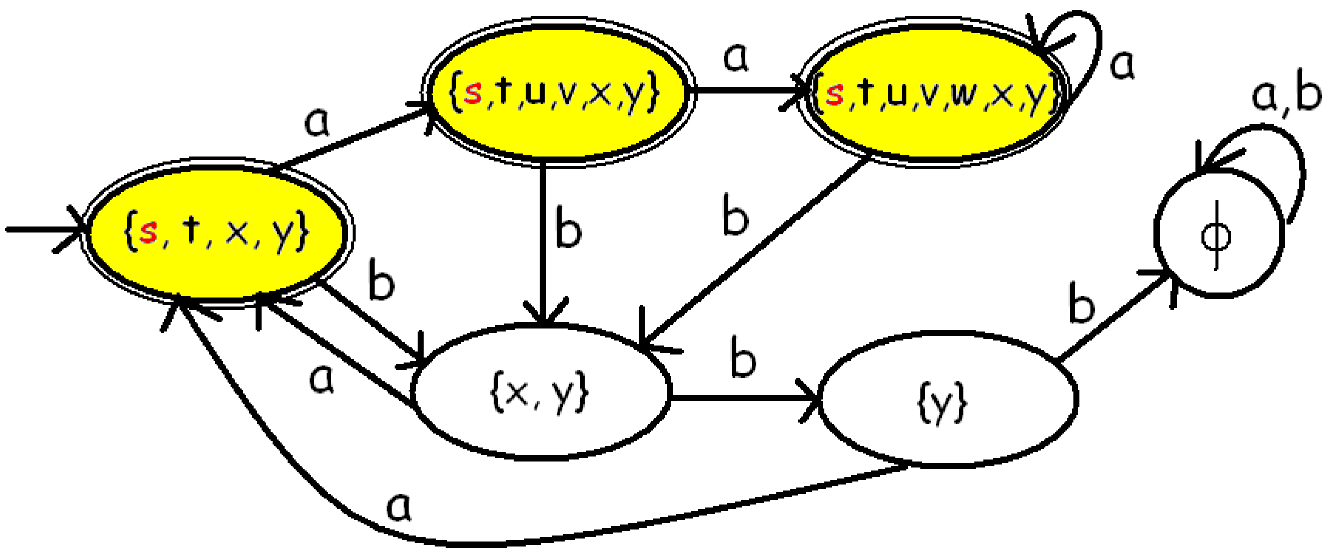
Your answer should generate the following strings but does not
aabaa, baaba, aaabaa, aabaaa, aabaab, abaaba, baaaba, baabaa,
baabab, bbaaba, aaaabaa, aaabaaa, aaabaab, aabaaaa, aabaaab,
aabaaba, aabaabb, abaaaba, abaabaa, abaabab

Lesson | Syntax | Hint | Answer

Previous Question | Next Question | Submit

# CSC 320 Lecture 9: Converting NDFA's to DFA's



Two machines $M_1$ and $M_2$ are equivalent if $L(M_1) = L(M_2)$.

Theorem: For any NDFA, there exists an equivalent DFA.

Proof: By construction.

Proofs by construction are nice because they don't just tell us that an object exists- they also give us an algorithm for constructing the object.

$E(q)$ = { p: p is reachable from q by following only transitions on ε}.

Note: q is always in $E(q)$.

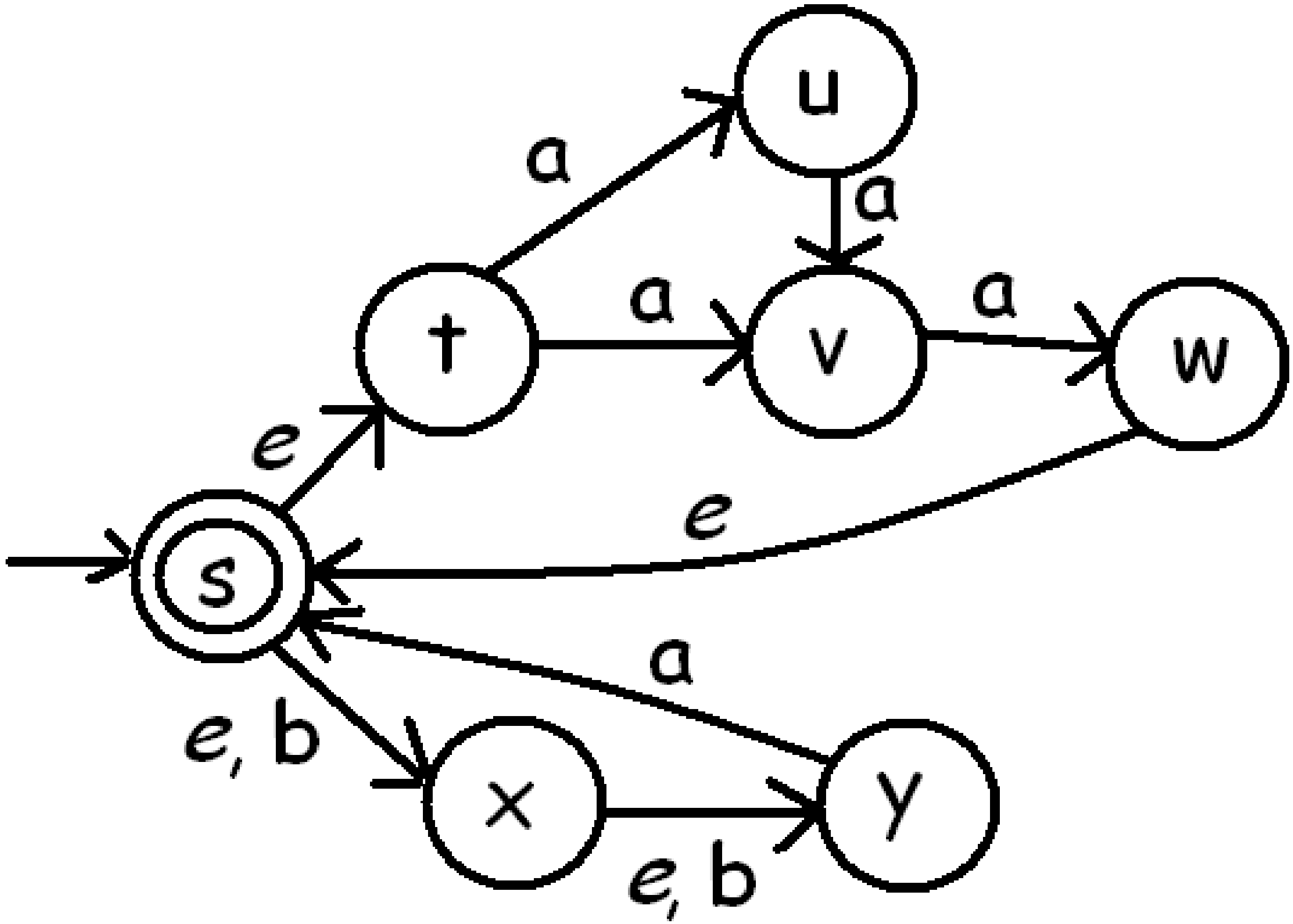
For a set S of states, $E(S) = \cup_{q \in S} E(q)$

Transition function for new DFA:

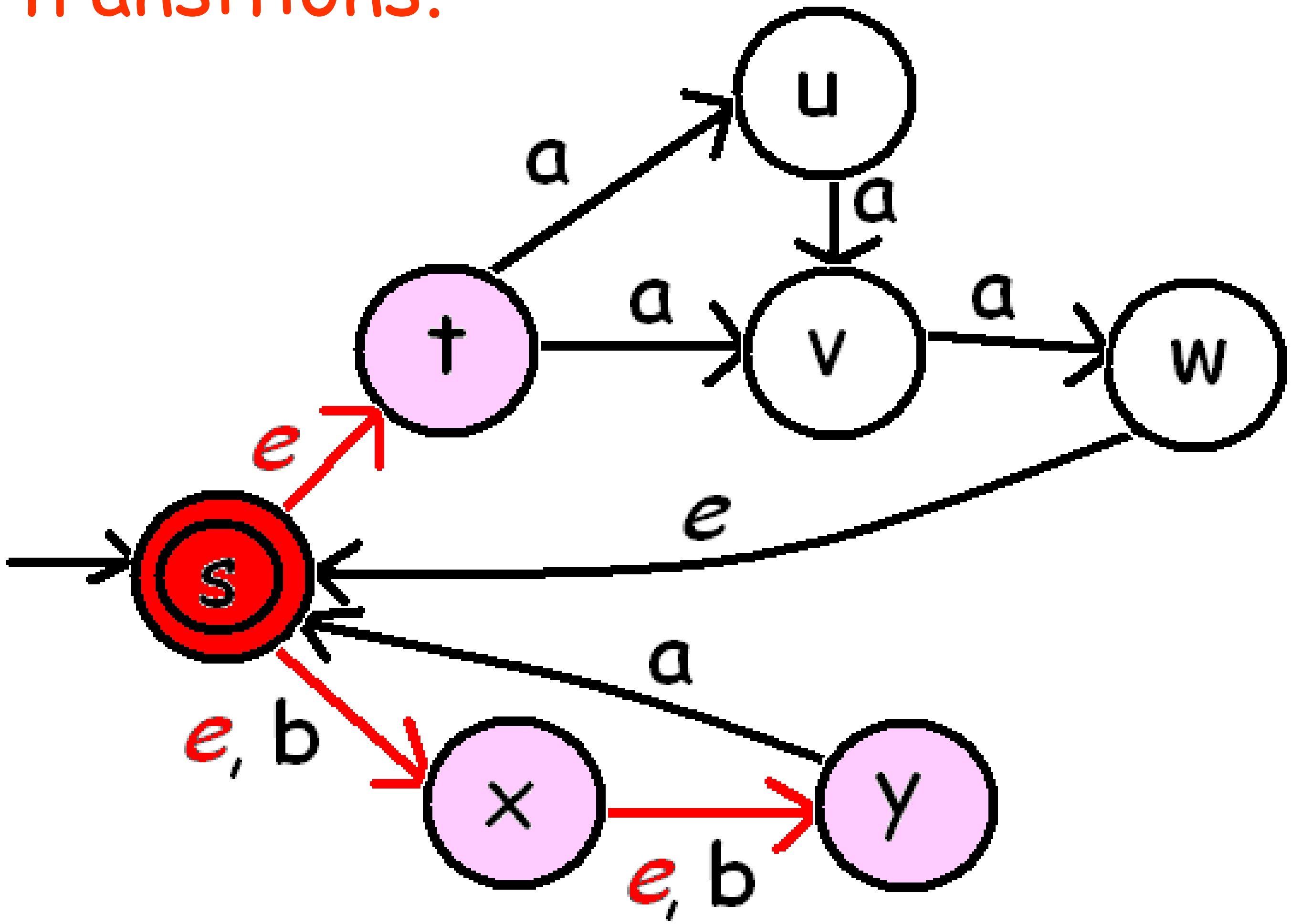
$\delta(P, \sigma) = E(Q)$ where

$Q = \{q$: for some p in P, $(p, \sigma, q)$ is in $\Delta\}$

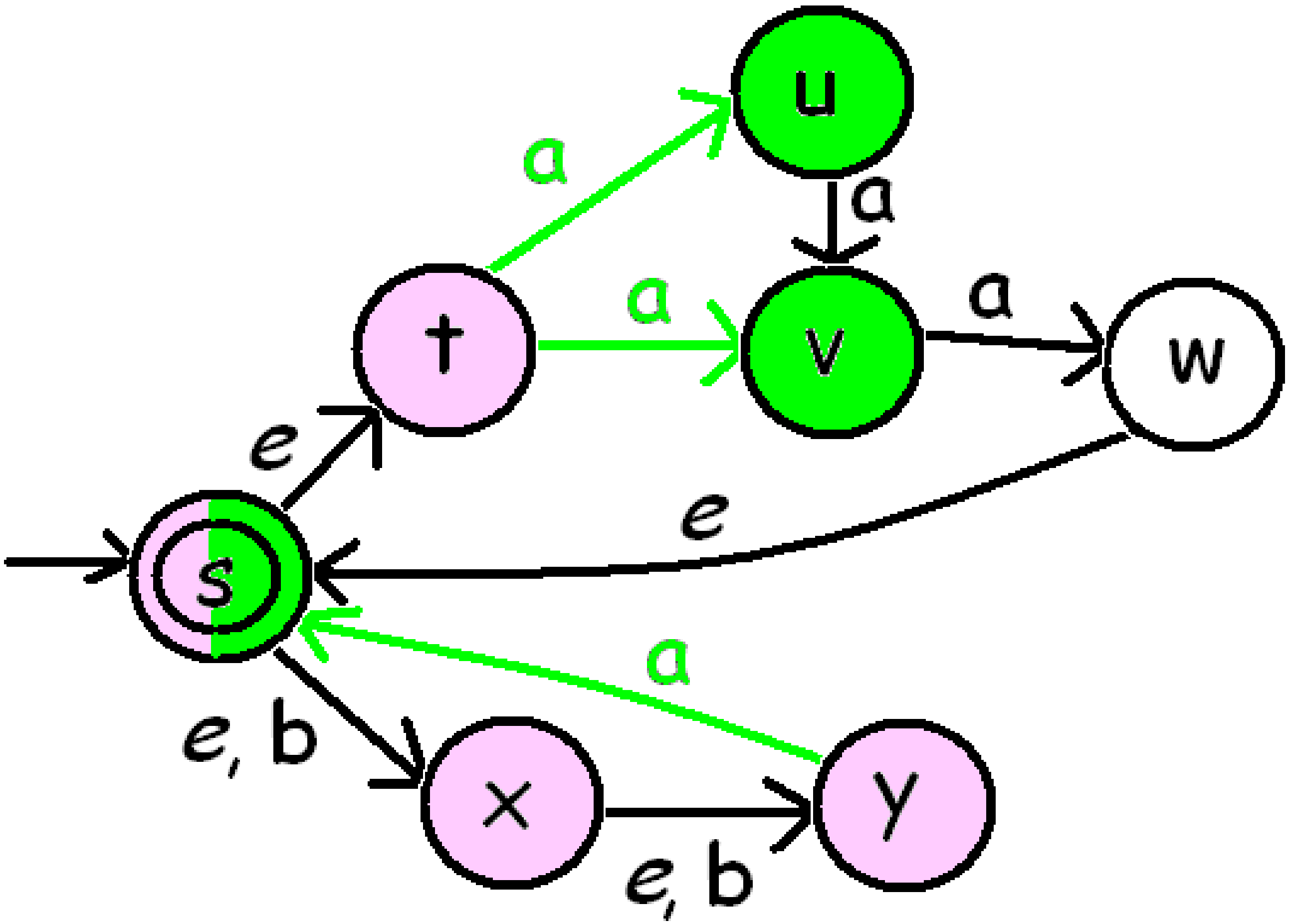
# Convert this NDFA to a DFA:



Start state is {s, t, x, y} = states reachable from s by traversing 0 or more e-transitions.

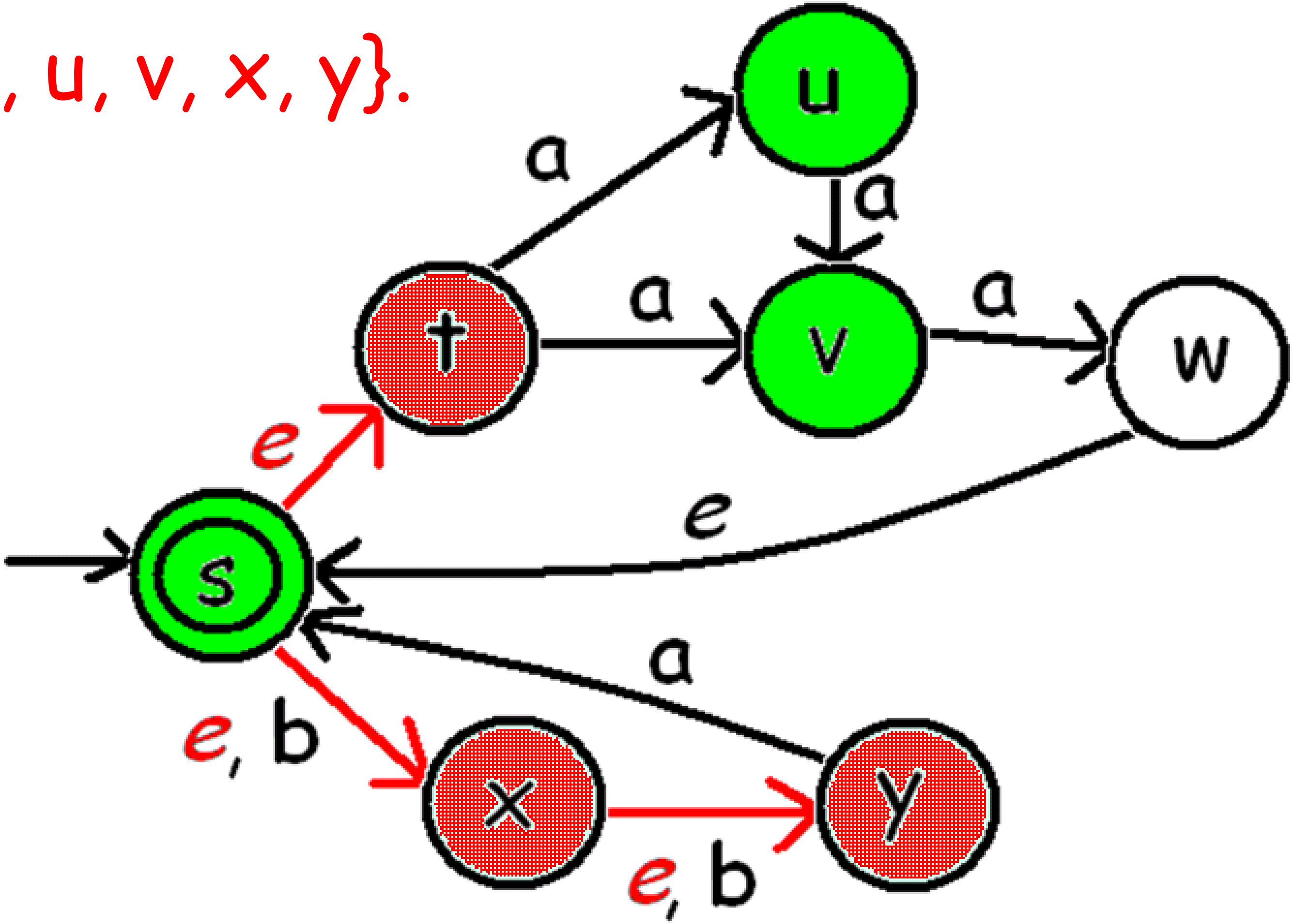


Read a from {s, t, x, y} and the next state can be {s, u, v}.

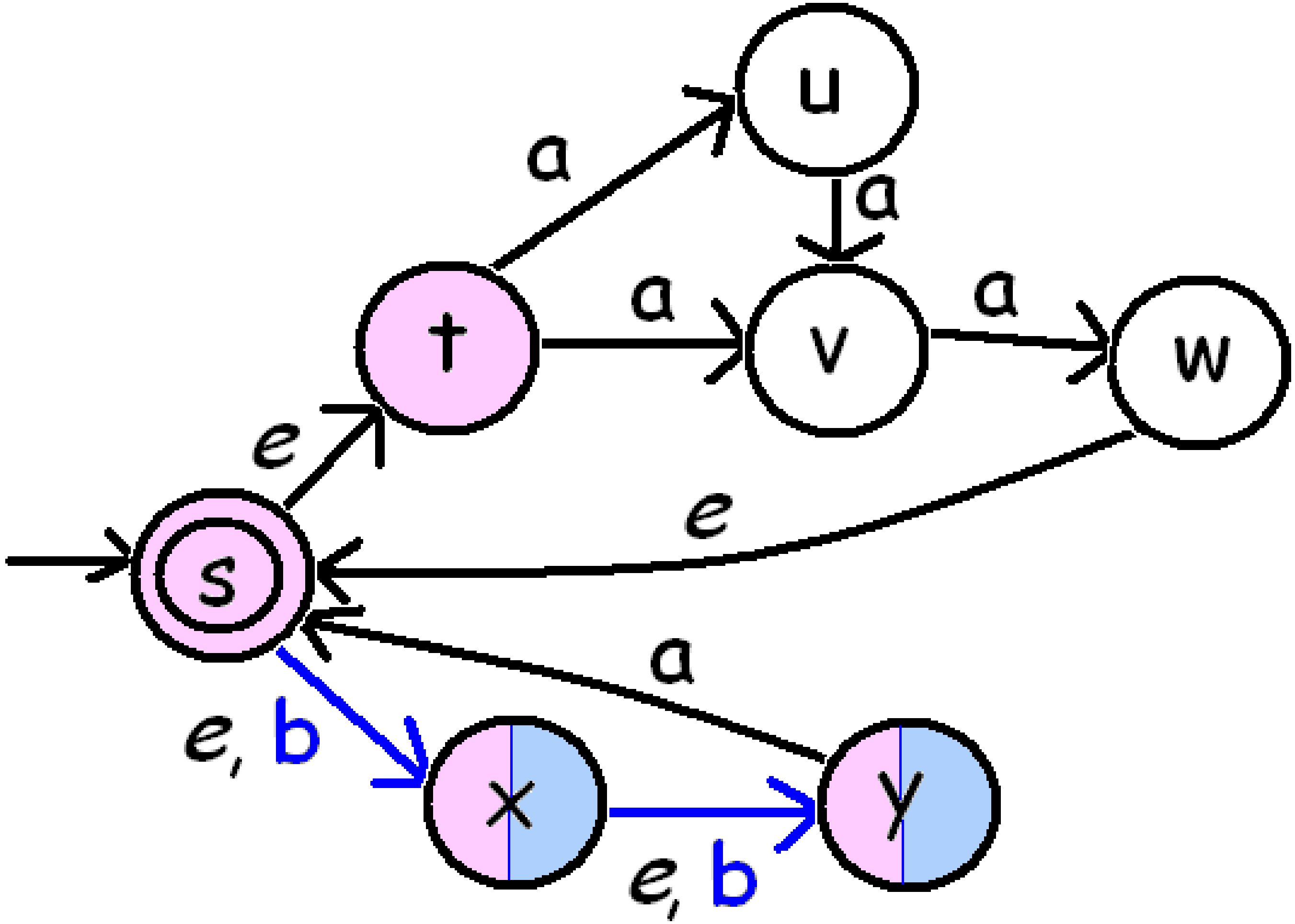


Then follow zero or more e-transitions from {s, u, v}, The next state is

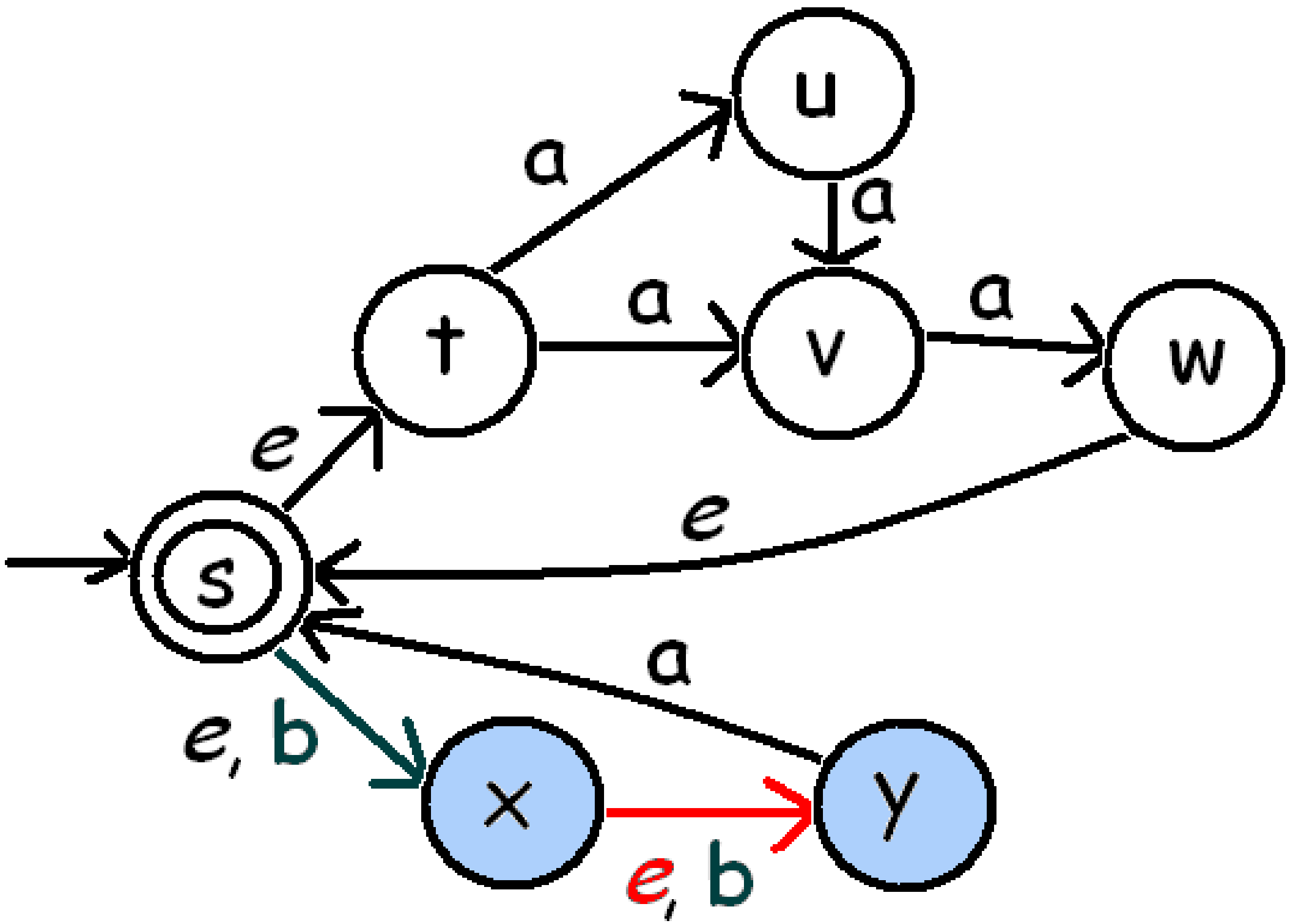
{s, t, u, v, x, y}.



Read b from {s, t, x, y} and the next state can be {x, y}.



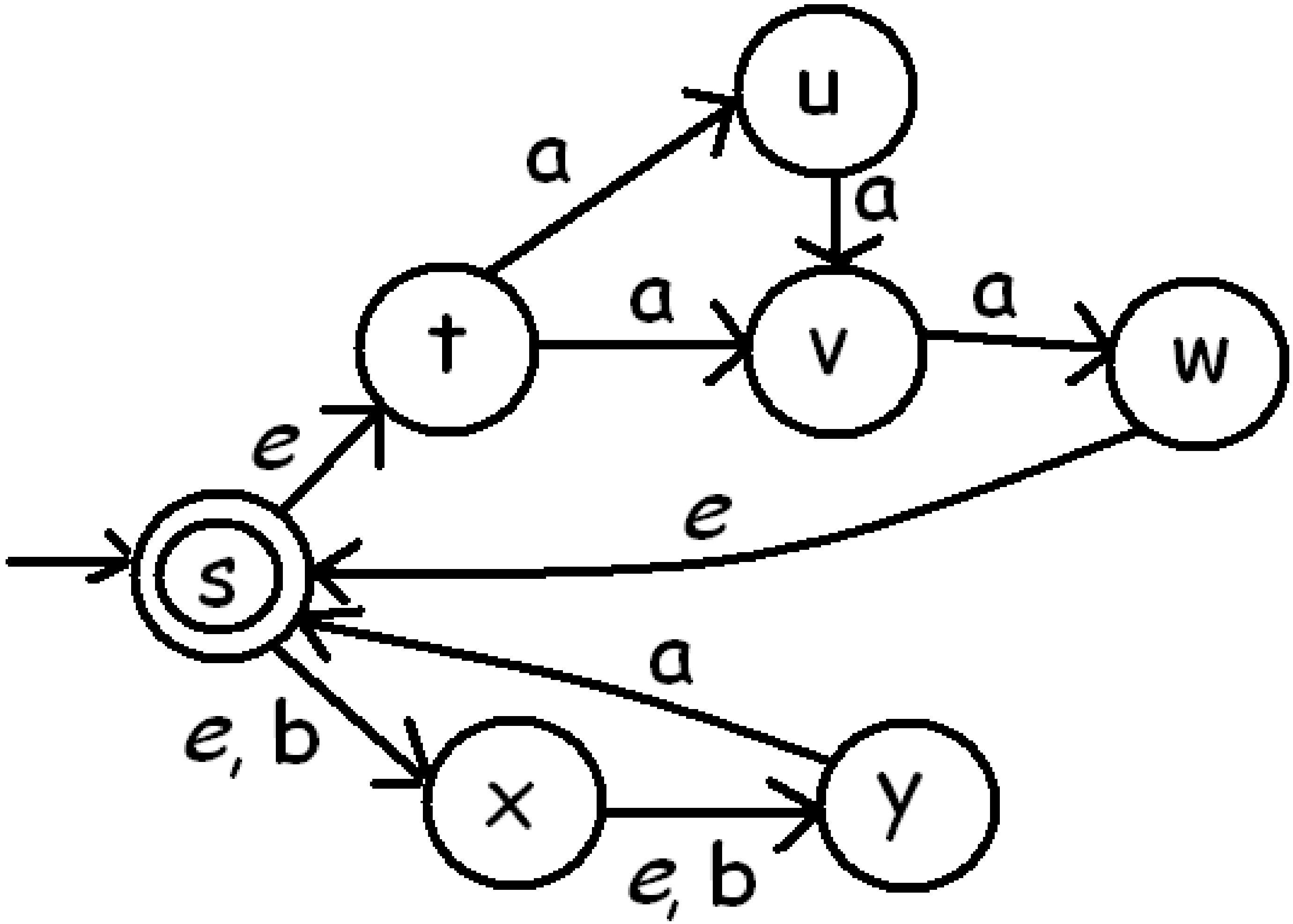
Then follow zero or more e-transitions from {x, y}, The next state is {x, y}.



The number of possible states could be exponential. But on assignments and exams, only a small subset of them will ever be pertinent.

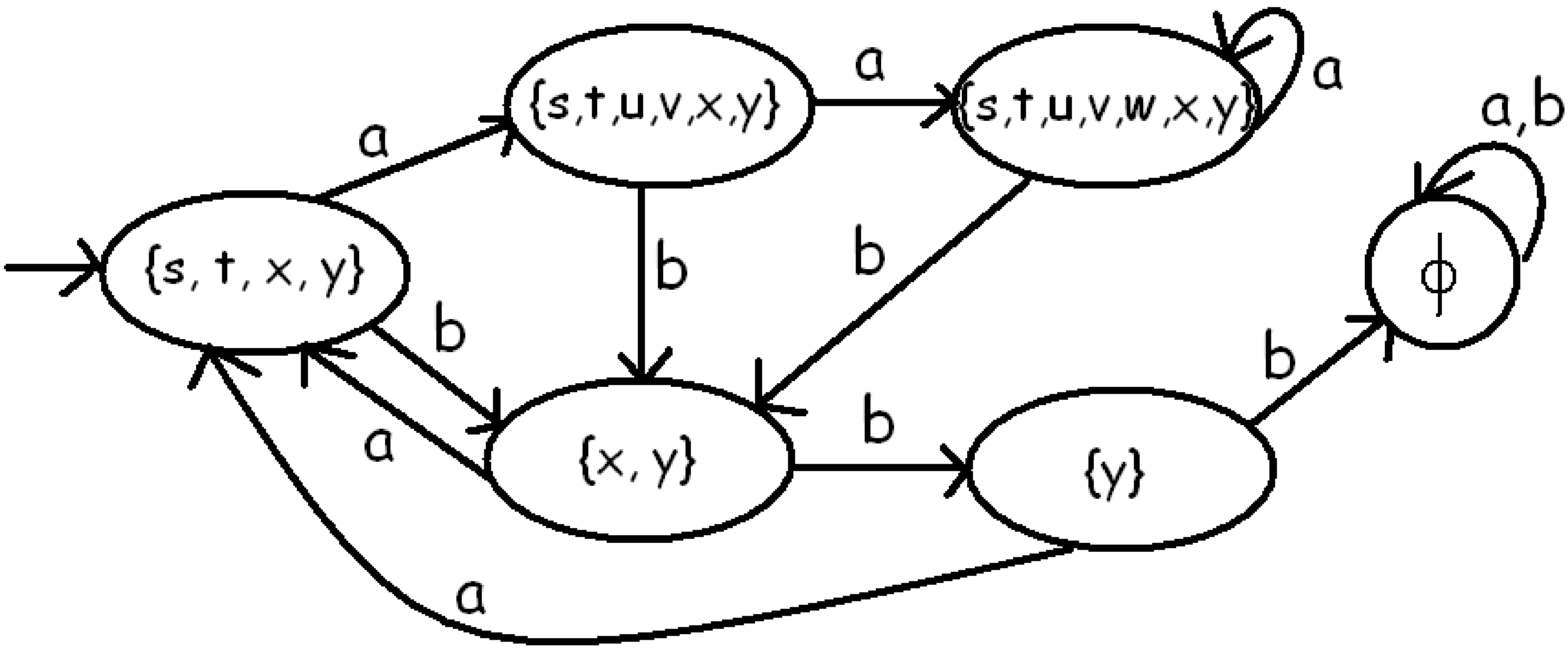
Only work out transitions for pertinent states.

# Convert this NDFA to a DFA:



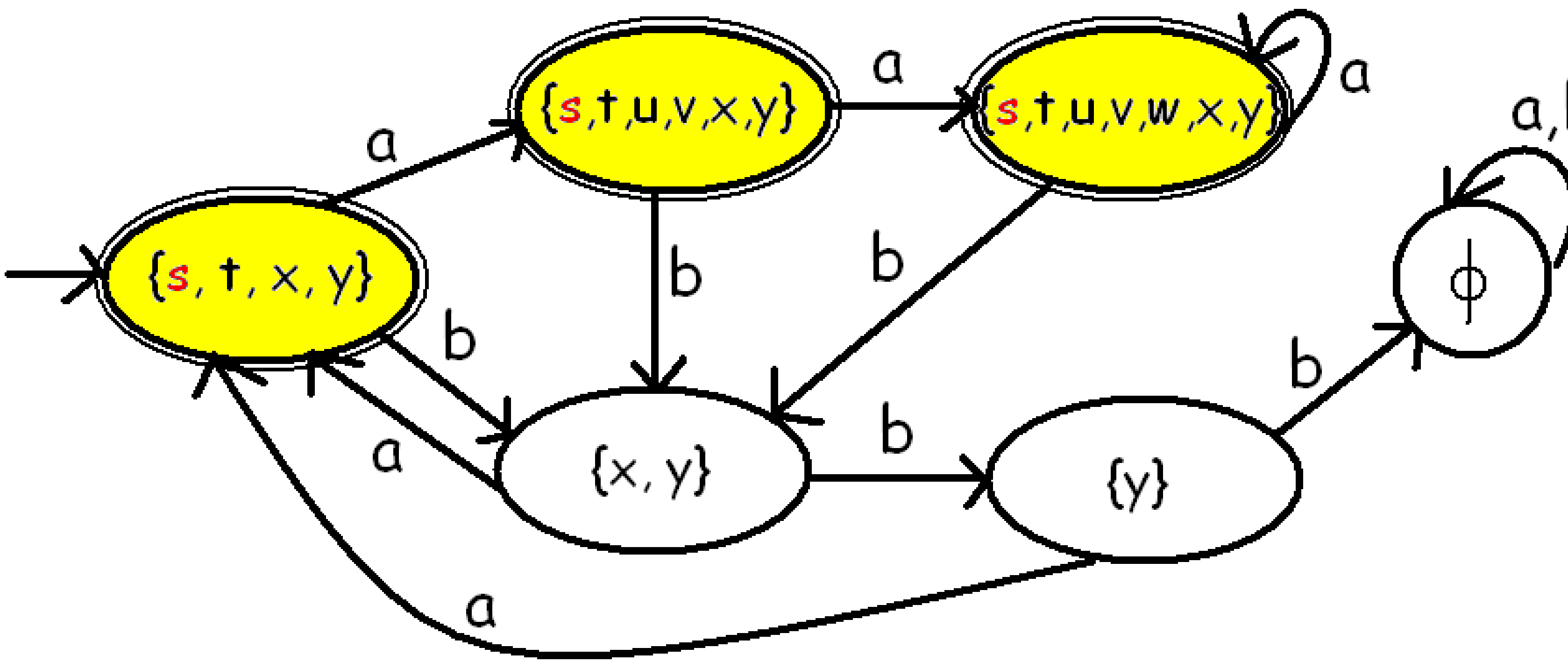
Which states should be final states?

Original machine: only final state was s.

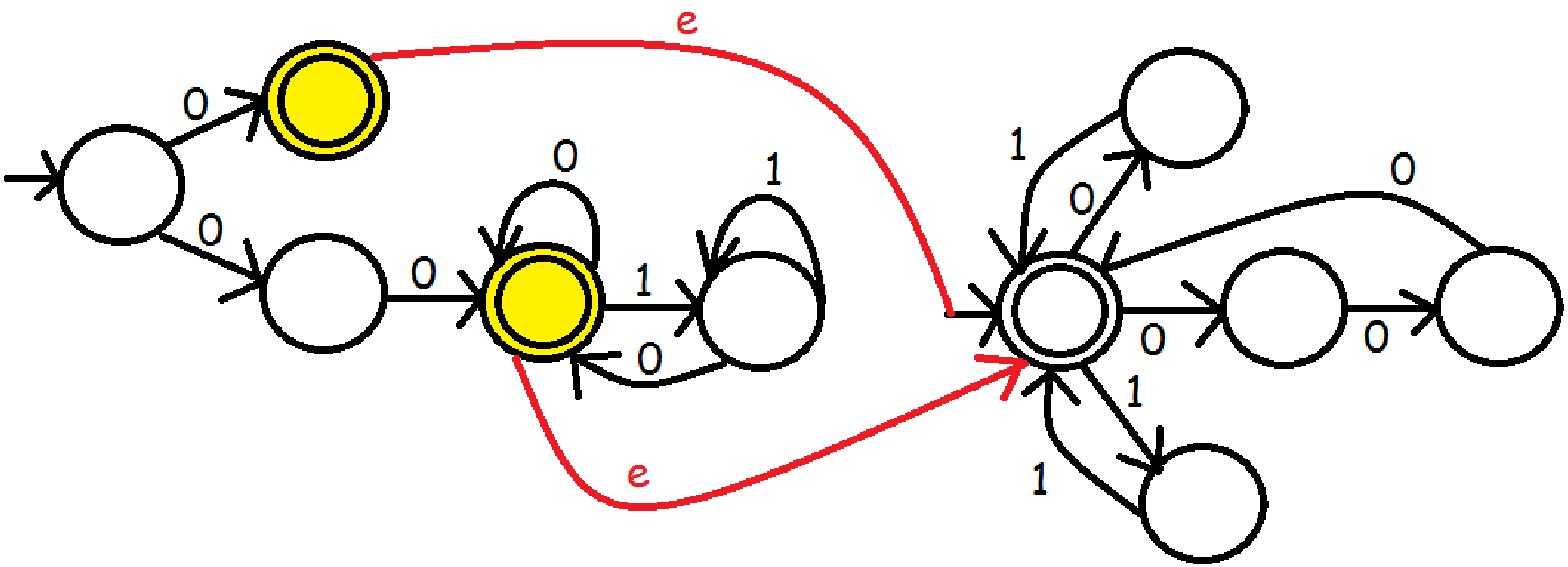


Which states should be final states?

Answer: new states whose subsets contain a state which was a final state originally.



{ w : w starts and ends with 0} •(000 U 11 U  01)*




This is correct because e is in the second language but in the general construction, we should change the yellow states to be non-final.