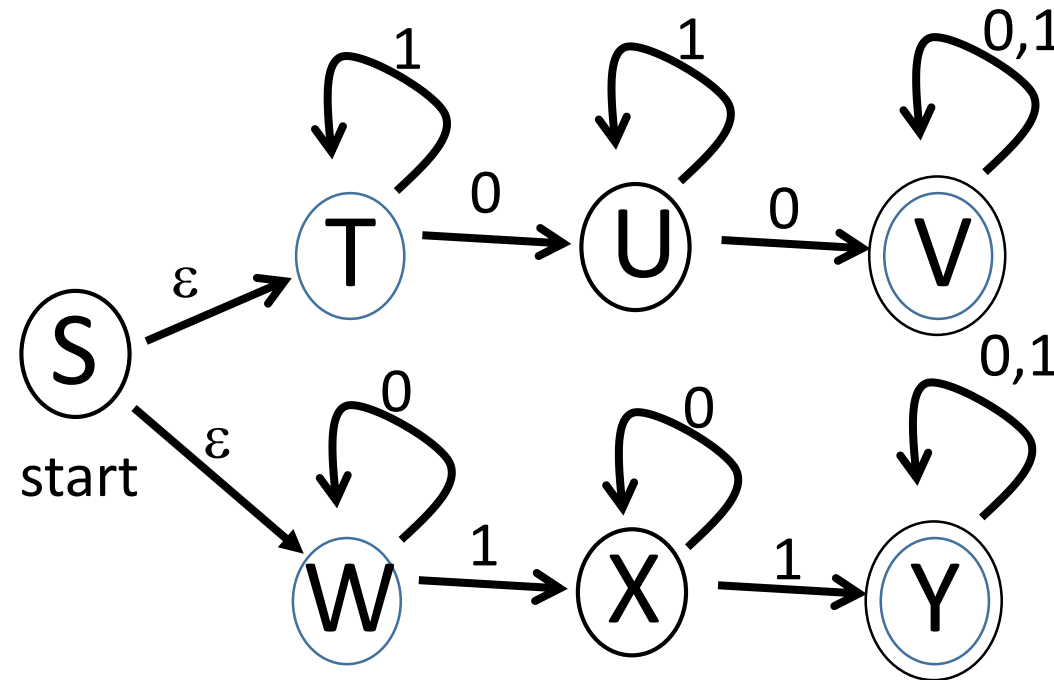


# $\epsilon$ -NFAs

See Section 2.5 of the text.

Here is another finite automaton -- an  $\epsilon$ -NFA, or an NFA with  $\epsilon$ -transitions". These allow transitions labeled " $\epsilon$ " to be followed without consuming any input. These aren't interesting in themselves but are useful for showing that DFAs and regular expressions describe the same languages.

Example: Here is an e-NFA that accepts strings with 2 0's or 2 1's:



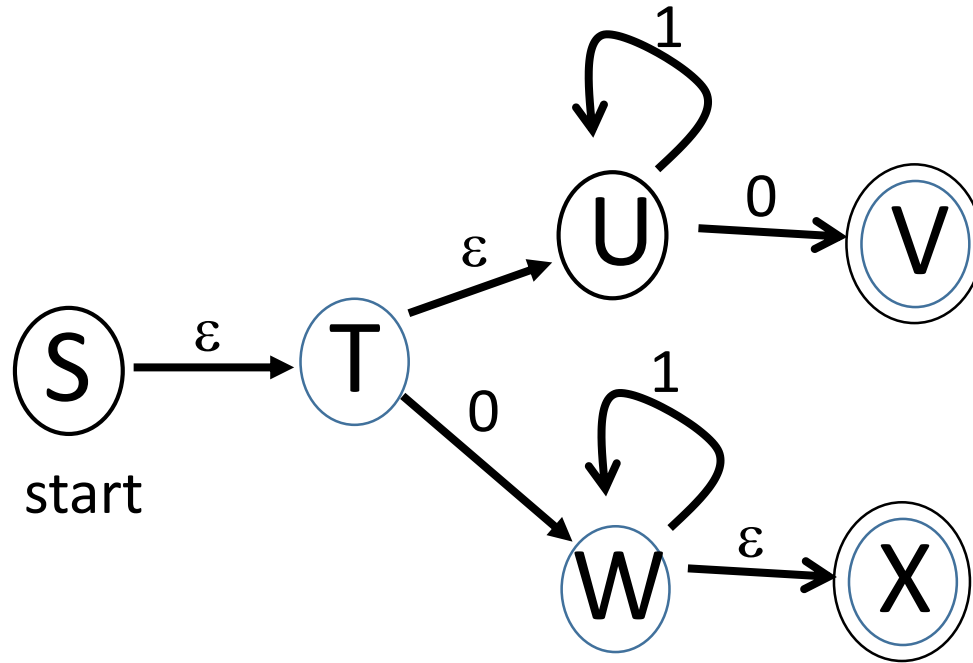
Formally, an  $\varepsilon$ -NFA is  $(Q, \Sigma, \delta, s, F)$  where  $Q$ ,  $\Sigma$ ,  $s$ , and  $F$  are defined as with other NFAs and the inputs to  $\delta$  are a state and either a letter in  $\Sigma$  or  $\varepsilon$ . This processes strings in the same way as the NFA  $(\Sigma, Q, \delta', s, F)$ , where  $\delta'(q,a)=\delta(q,a) \cup \bigcup_{q' \in \delta(q,\varepsilon)} \delta'(q', a)$ . (Note that this is a recursive definition of  $\delta'$ .)

We are going to show that the language accepted by an  $\varepsilon$ -NFA is regular (so every  $\varepsilon$ -NFA has an equivalent DFA). To get there we need the idea of an  $\varepsilon$ -closure of a set of states. Let  $(Q, \Sigma, \delta, s, F)$  be the  $\varepsilon$ -NFA we are talking about and let  $A$  be a set of states from  $Q$ .  $\bar{A}$  will represent the  $\varepsilon$ -closure of  $A$ . Here are two things we want to be true:

- $A \subset \bar{A}$
- For each  $q$  in  $\bar{A}$  if there is an  $\varepsilon$ -transition from  $q$  to  $q_1$  then  $q_1$  should be in  $\bar{A}$ .

This gives us an algorithm: to compute  $\bar{A}$  start with  $A$  and add the destinations of  $\varepsilon$ -transitions until nothing else can be added.

Example:



This accepts  $1^*0+01^*$  Here are some  $\epsilon$ -closures:

$$\overline{\{T\}} = \{T, U\}$$

$$\overline{\{U\}} = \{U\}$$

$$\overline{\{S\}} = \{S, T, U\}$$

$$\overline{\{S, W\}} = \{S, W, T, U, X\}$$

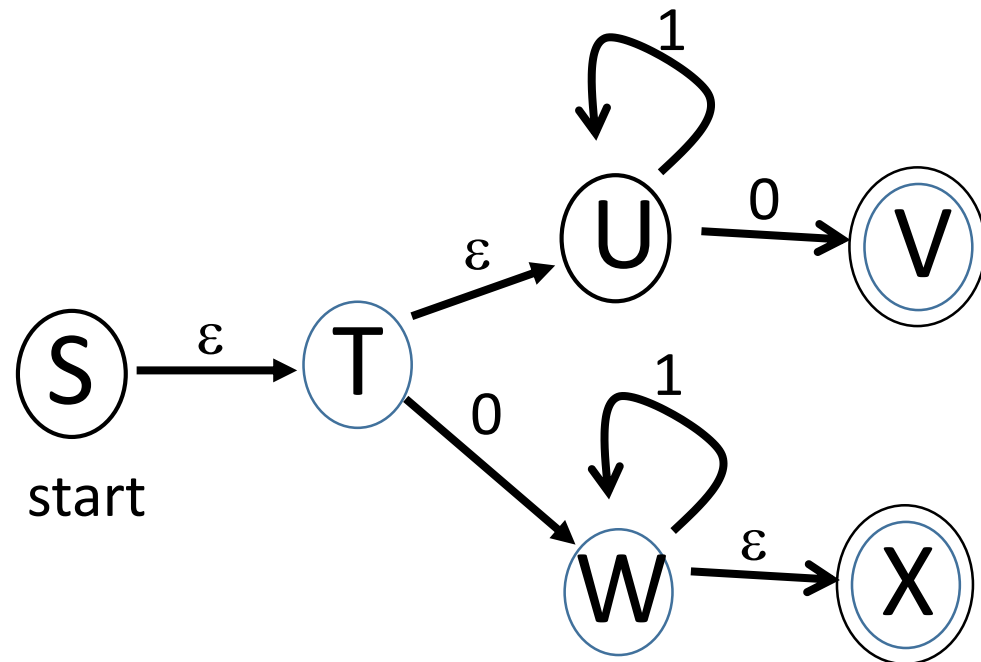
**Theorem:** Any language accepted by an  $\varepsilon$ -NFA is regular.

**Construction:** Let  $(Q, \Sigma, \delta, s, F)$  be an  $\varepsilon$ -NFA. We will construct an equivalent DFA  $(Q', \Sigma, \delta', s', F')$  :

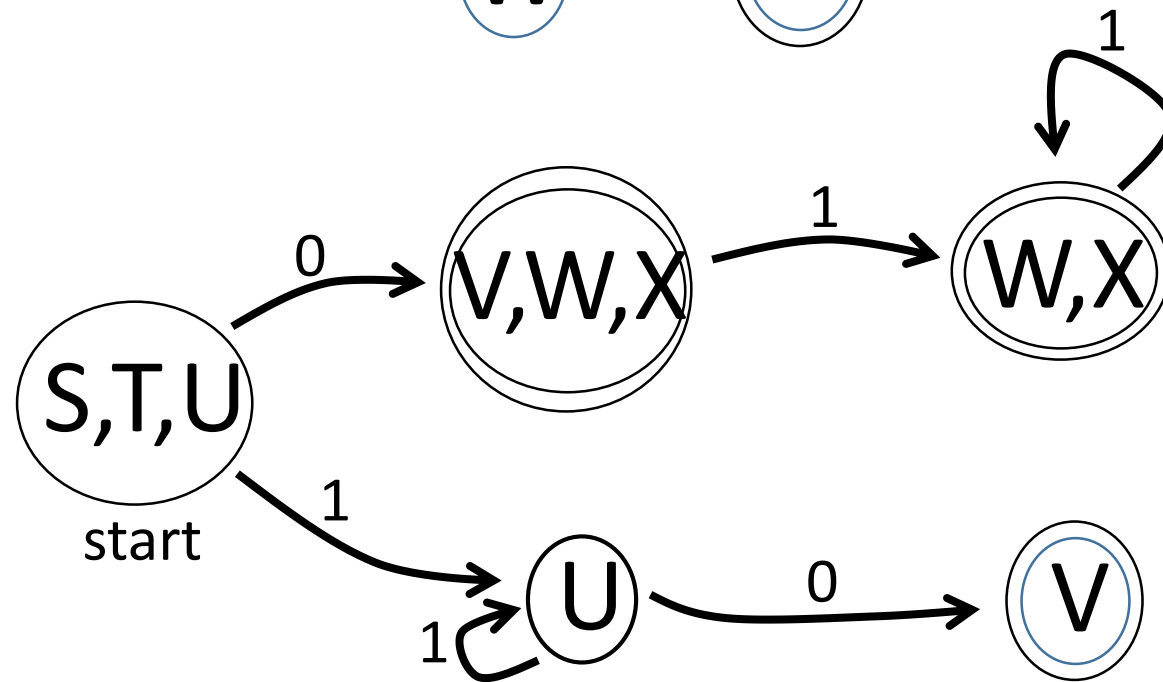
- $Q'$  consists of subsets of  $Q$
- $s' = \overline{\{s\}}$
- If  $P = \{q_0, q_1, \dots, q_k\}$  is a state in  $Q'$  and  $a$  is in  $\Sigma$  then  $P' = \overline{\bigcup_{i=0}^k \delta(q_i, a)}$  is also a state in  $Q'$  and  $\delta'(P, a) = P'$
- If  $P = \{q_0, q_1, \dots, q_k\}$  is a state in  $Q'$  and if any of the  $q_i$  are in  $F$ , then  $P$  is in  $F'$ .

Example:

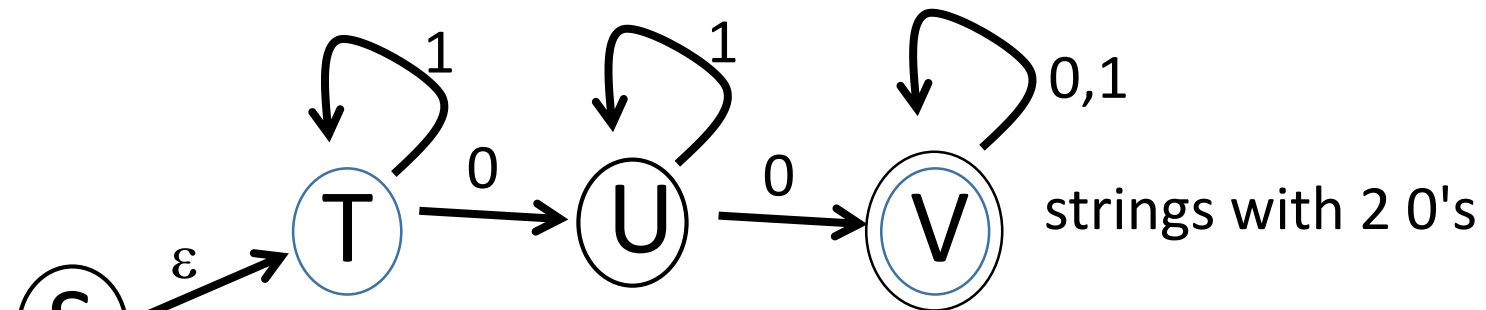
$\epsilon$ -NFA:



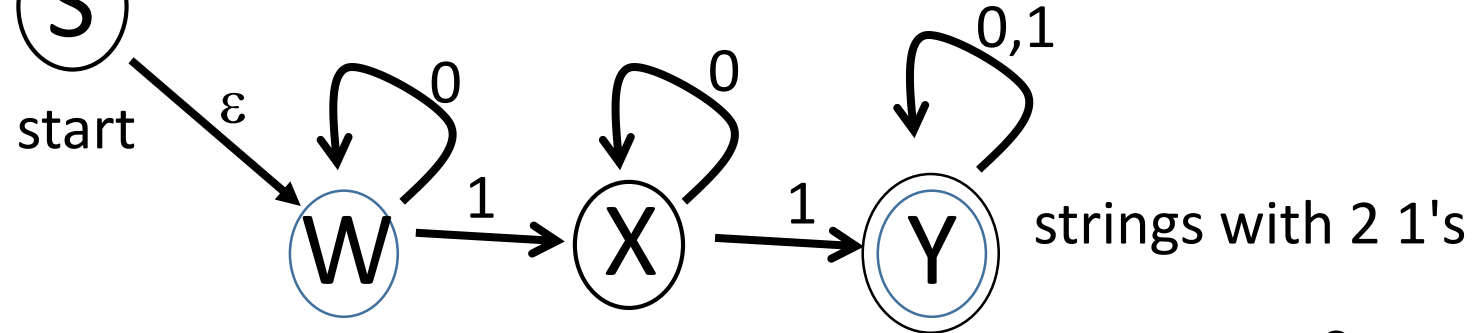
Equivalent DFA:



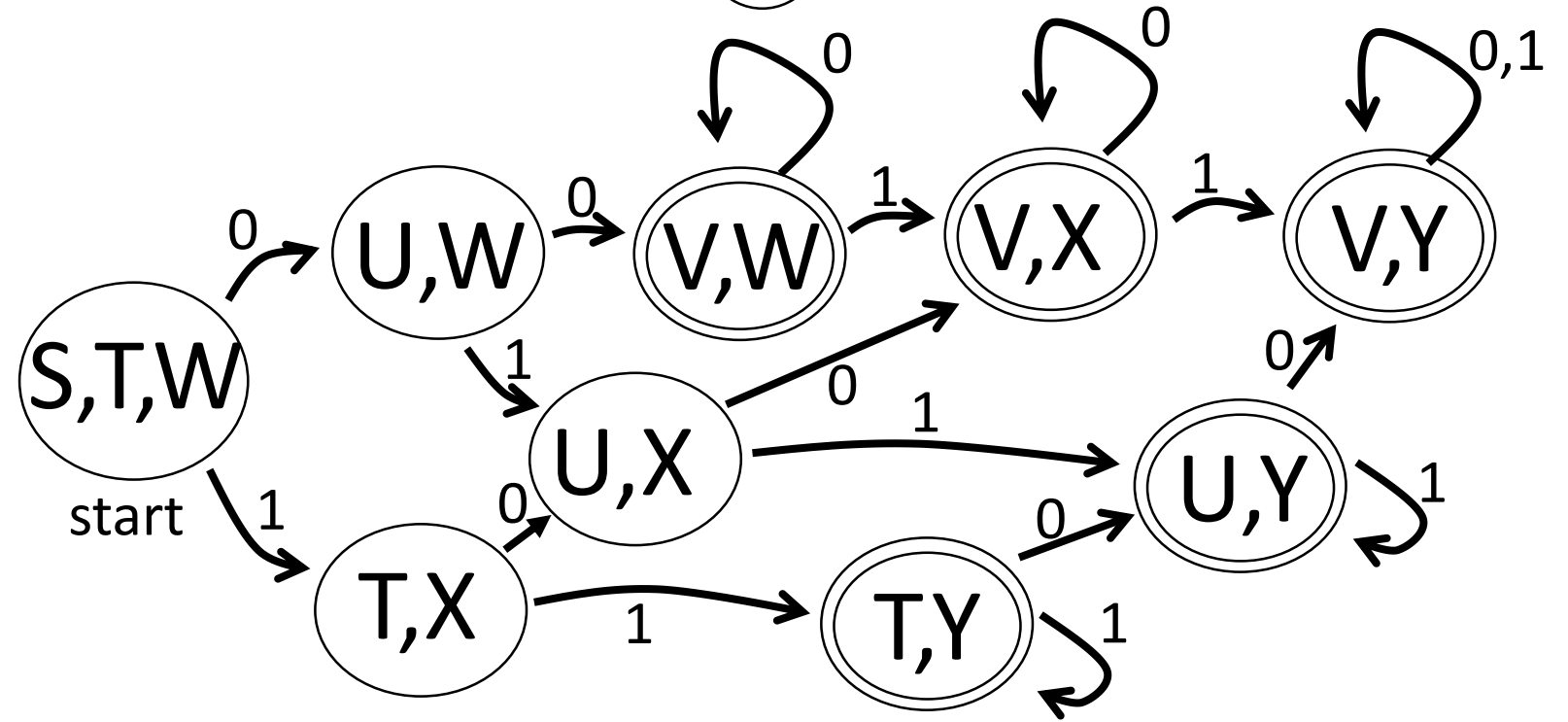
Example:



$\epsilon$ -NFA:

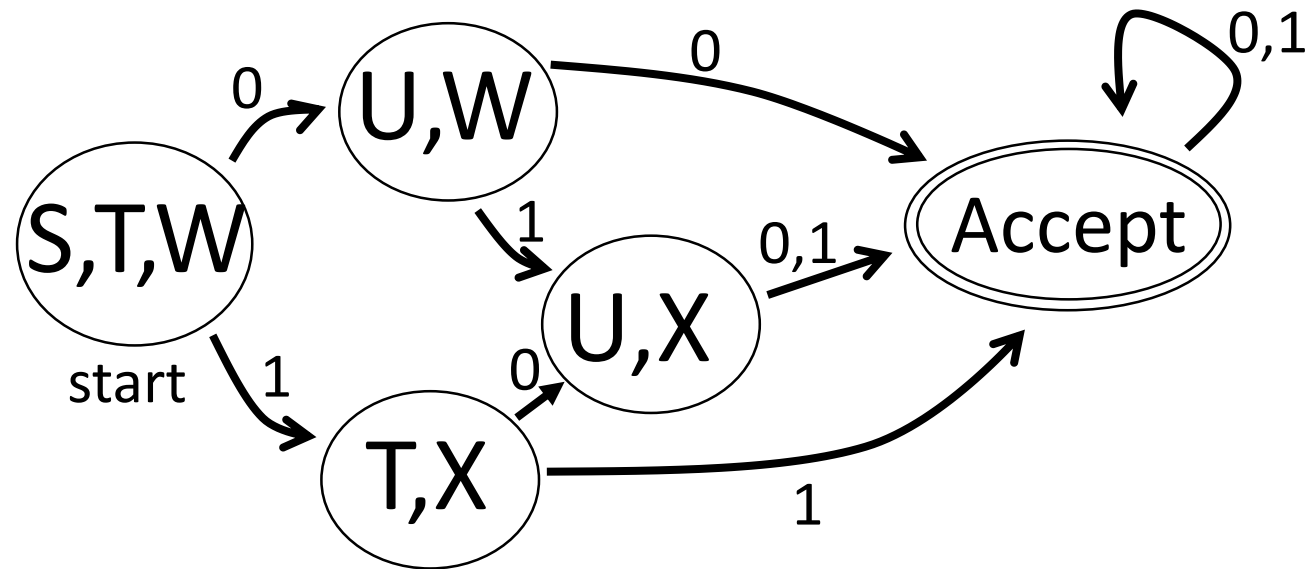


Equivalent DFA:





Note that this can be simplified to:



1

We still need to prove that the DFA of this construction accepts exactly the same strings as the original  $\varepsilon$ -NFA. The proof is almost exactly the same as the proof that NFAs are equivalent to DFAs. If a string is accepted by the  $\varepsilon$ -NFA, processing the string takes the automaton through states  $q_0, q_1, \dots, q_k$ , where  $q_k$  is final. The  $q_i$  will be elements of states through which the DFA will pass while processing the string. The DFA will end in a state containing  $q_k$ , which is final, so it will accept the string.

Alternatively, if a string  $\alpha$  takes the DFA to state  $\{q_0, \dots, q_j\}$  then on input  $\alpha$  the  $\varepsilon$ -NFA could be in any of the  $q_i$  states. If the string is accepted by the DFA it will also take the  $\varepsilon$ -NFA to an accept state.