Equivalence

See section 4.4 of the text

Our algorithm for converting a regular expression into a DFA produces a correct but overly complex automaton. We will now look at some technique for simplifying automata.

First, we'll say that a state P in a DFA is *reachable* if there is some string that takes the automaton from the start state to P.

Note that

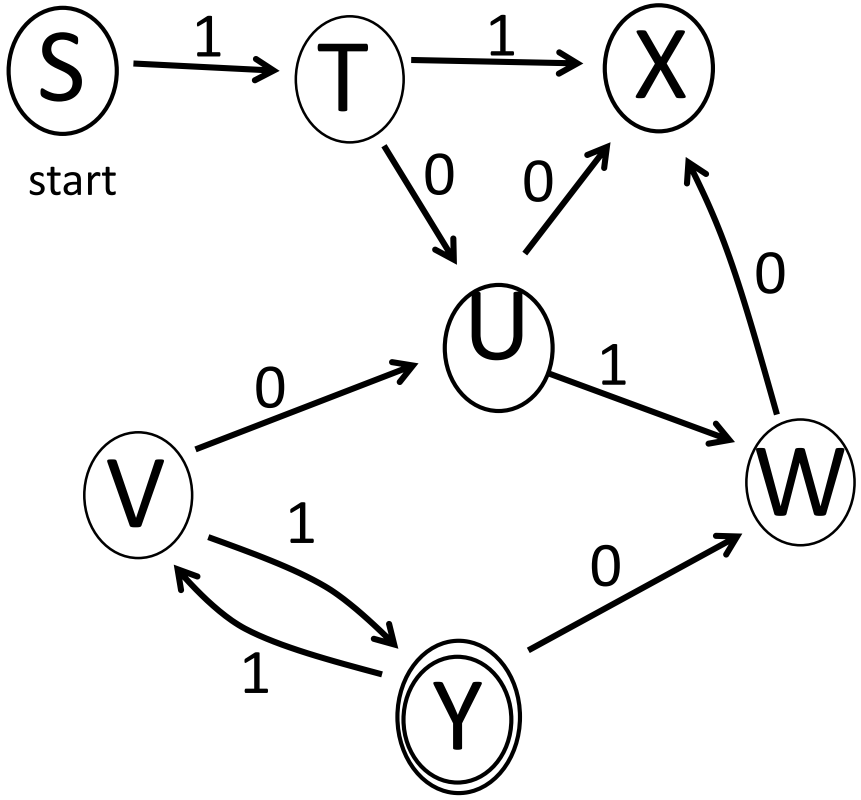
- The start state is reachable.
- If state P is reachable and there is a transition $\delta(P,a)=Q$, then state Q is also reachable.

Here is a marking algorithm for finding reachable states:

- Mark the start state.
- Repeat the following until nothing new can be marked:
  - If P is marked and there is a transition $\delta(P,a)=Q$, then mark Q.

In the end, any state that is marked is reachable.

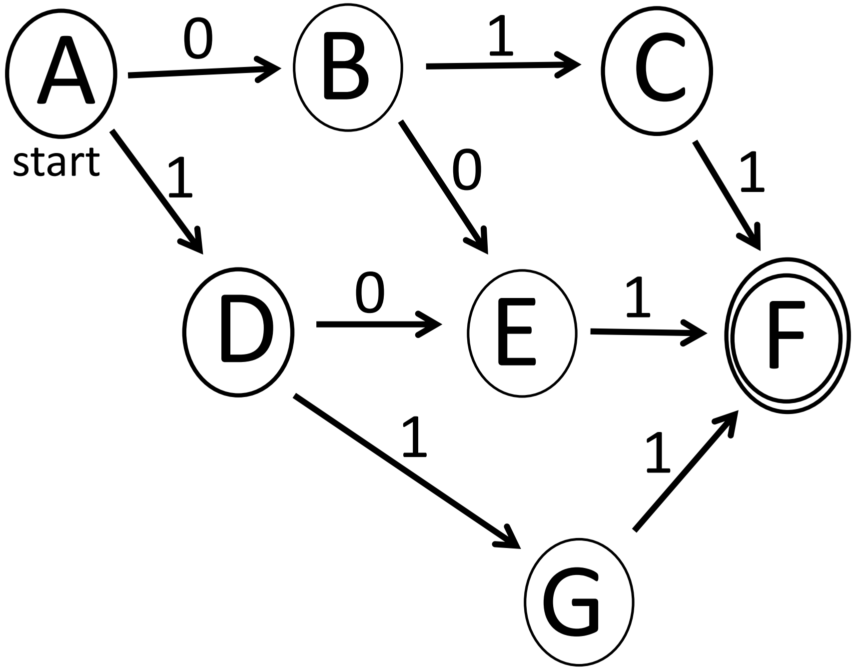
Example:



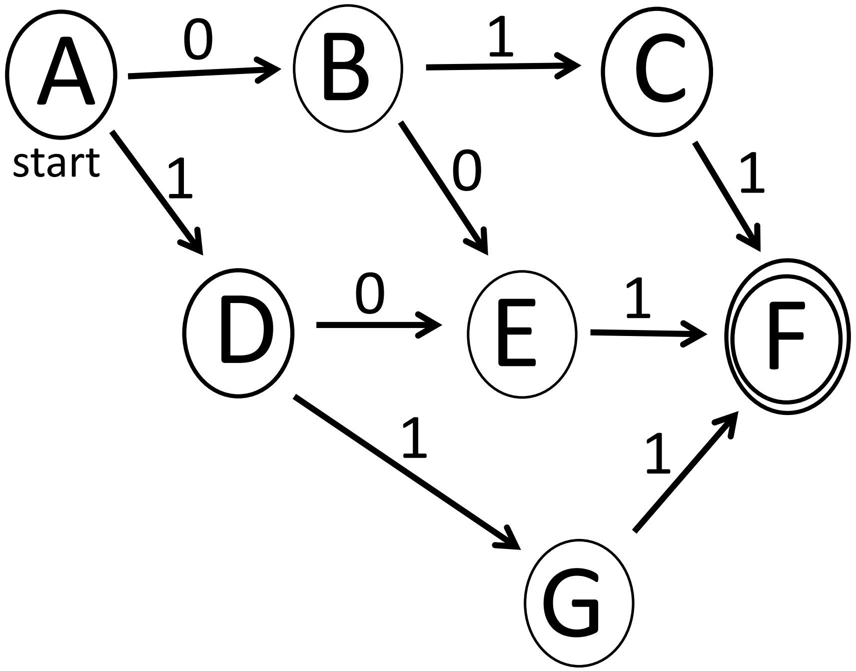
We mark state S, then T, then U and X, then W. There are no transitions from any of these states to any new state, so the algorithm ends. This leaves V and Y unmarked, so they are unreachable. Since there is no reachable final state, the automaton accepts no strings.

We say states p and q are *equivalent* if every string that takes p to a final state also takes q to a final state, and vice-versa.

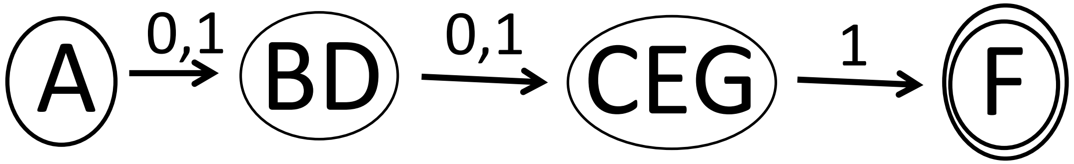
Example:



States B and D are equivalent because from either we can get to a final state with strings 01 and 11. Similarly, states C, E, and G are all equivalent.



Since B and D are equivalent, and so are C,E, and G, we can rewrite this automaton as

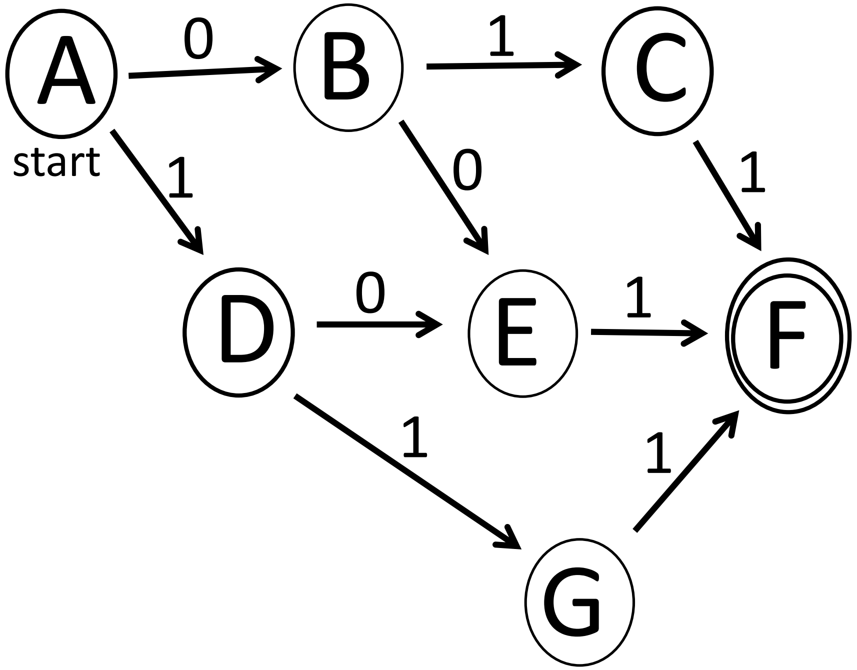


Algorithm for finding equivalent states: Make a table of all (unordered) pairs of states. We will mark all pairs that are *not* equivalent, so at the end any unmarked pairs are equivalent.

A. If state p is final and q is not, mark (p,q).
B. If (p,q) is marked and (p_1,q_1) is another pair of states so that for some a $\delta(p_1,a)=p$ and $\delta(q_1,a)=q$, then mark (p_1,q_1).

Continue this until nothing more can be marked.

Example:



Initially we mark every pair containing F.

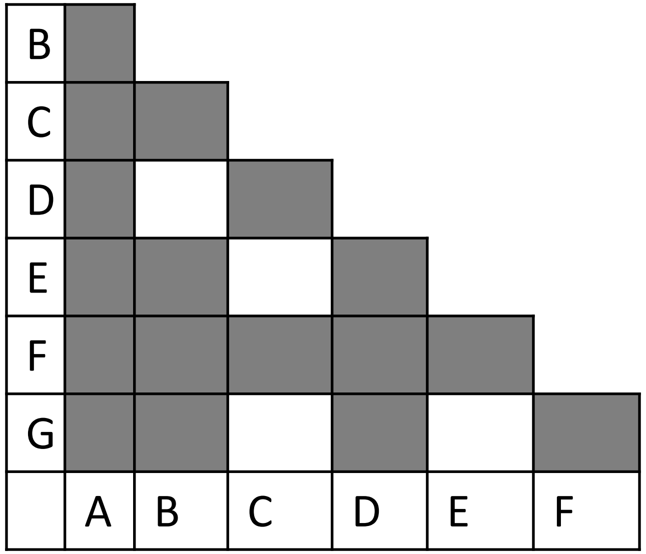
(C,F)=> (B,E), (B,G), (B,C)

(G,F)=> (D,G), (D,C), (D,E)

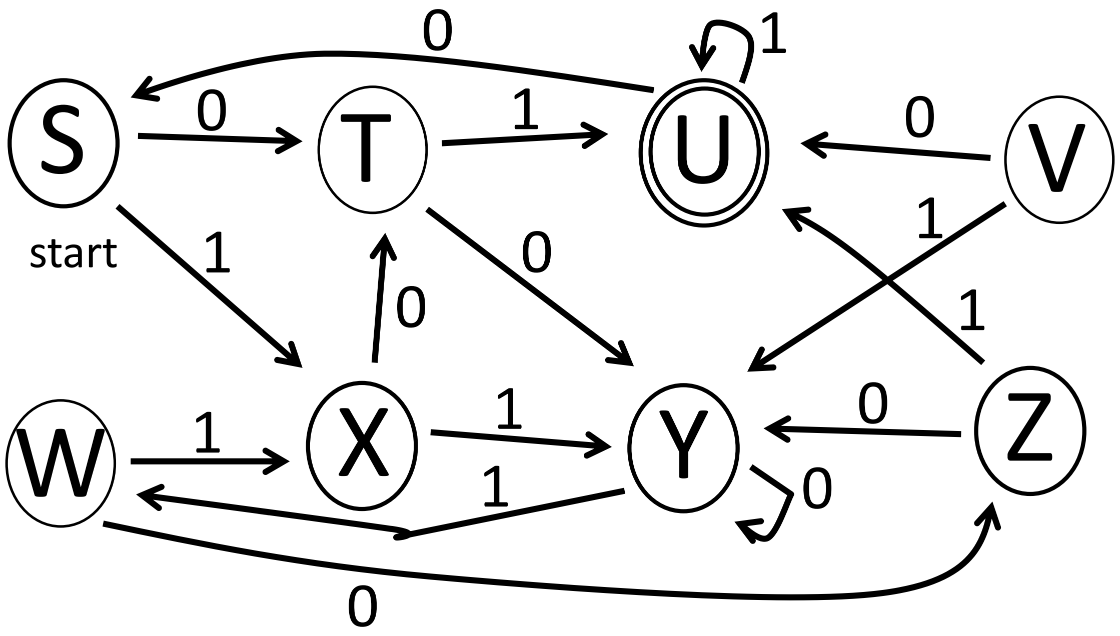
(D,F)=> (A,E), (A,C), (A,G)

(B,E)=> (A,D)

(D,C)=> (A,B)



Example:



Initially mark every pair containing U.

(T,U)=> (S,V),(X,V)

(W,U)=>(Y,T),(Y,Z)

(X,U)=>(S,T),(W,T),(W,Z) (S,Z)

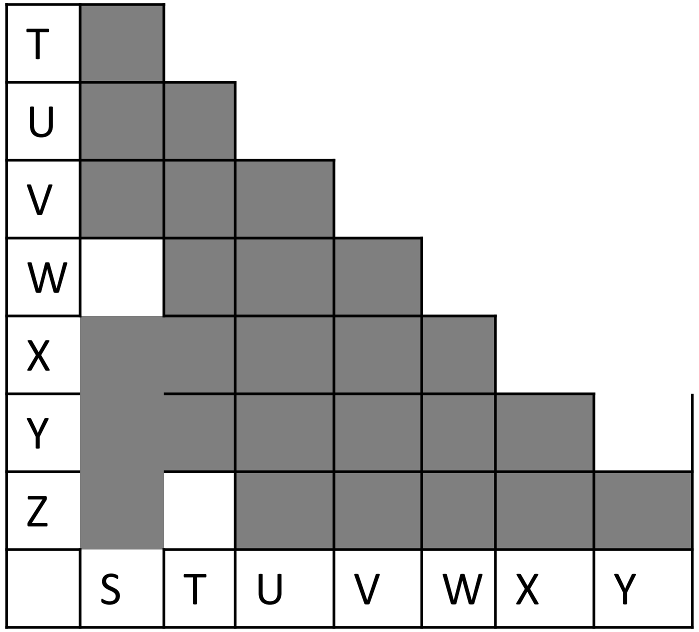
(Y,U)=> (T,V),(Z,V),(X,T), (X,Z), (V,Y)

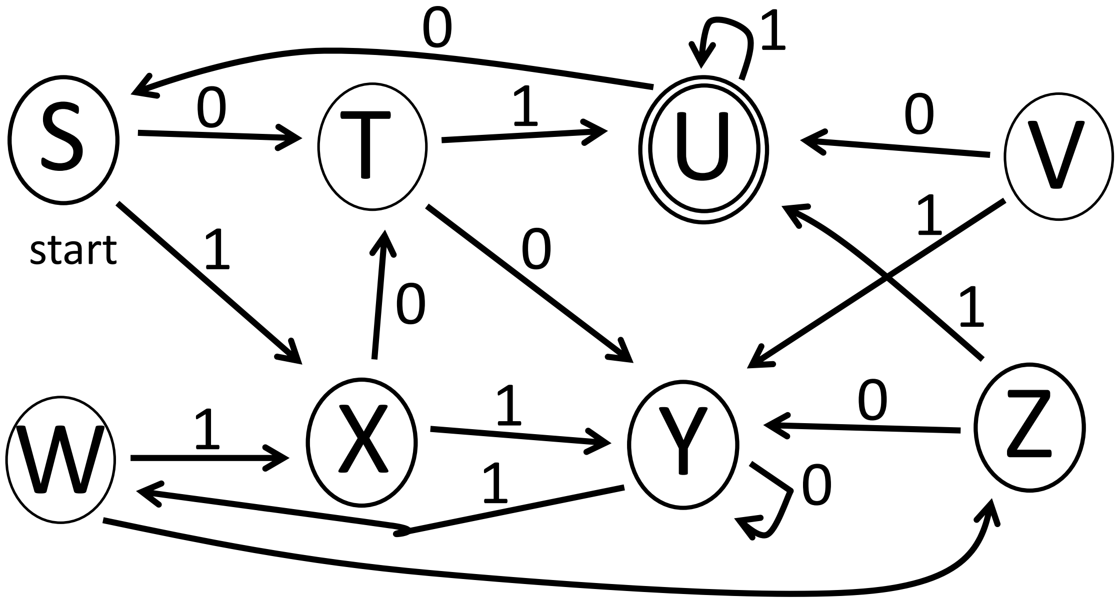
(Z,U)=> (W,V)

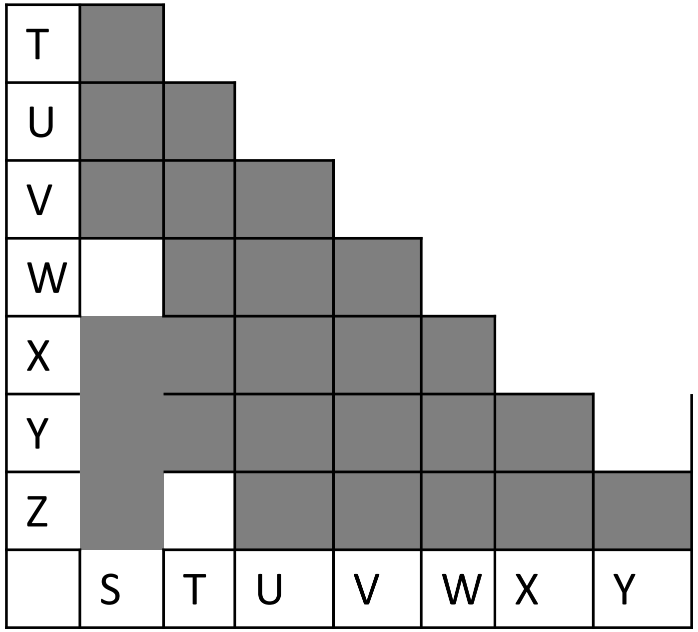
(Y,T)=> (X,Y),(S,Y)

(X,Y)=> (S,X), (W,X)

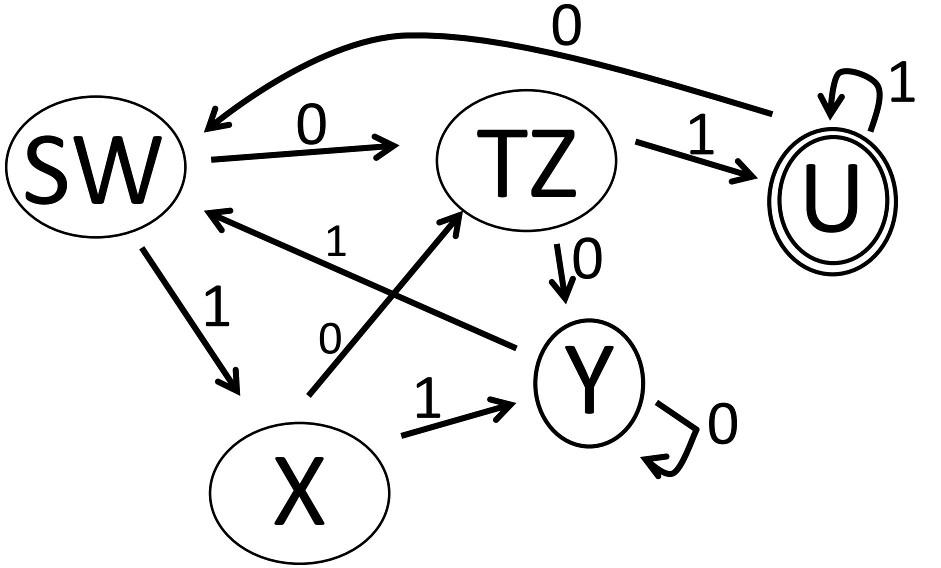
(W,X)=>(W,Y)





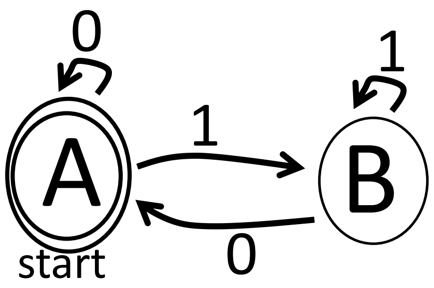


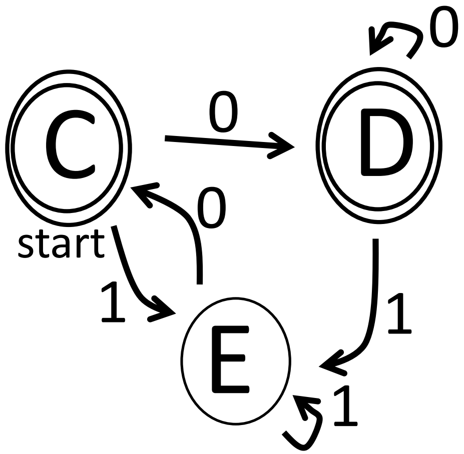
So (S,W) and (T,Z) are equivalent and V is unreachable. We can simplify the automaton to the one at right:



We can use this to test whether two DFAs accept the same language.
This will be true if their start states are equivalent.

For example:





B				
C				
D				
E				
	A	B	C	D

So {A,C,D} are equivalent and so are {E,B}
In particular the two start states are equivalent, so the DFAs accept the same language.

Here's an algorithm for finding a DFA with a minimum number of states equivalent to a given DFA:

1. Eliminate any state that can't be reached from the start state.
2. Partition the remaining states into blocks of equivalent states.
3. Rebuild the automaton using these blocks as the states. Note that if states P and Q are in the same block then for any a $\delta(P,a)$ and $\delta(Q,a)$ must be equivalent or else P and Q would not be equivalent. This lets us expand the transition function as a function between blocks.

Suppose we start with DFA $\mathcal{L}$ for some language and we use this algorithm to construct a "minimal" DFA $\mathcal{M}=(\Sigma,Q,\delta,s,F)$. Let $\mathcal{O}=(\Sigma,Q',\delta',s',F')$be any other DFA for the language. The start states s and s' of M and O must be equivalent because the two DFAs accept the same languages. Similarly, for any a in Σ, $s_1=\delta(s,a)$ and $s_1'=\delta'(s',a)$ must be equivalent, $s_2=\delta(s_1,a)$ and $s_2'=\delta'(s_1',a)$ are equivalent, and so forth. Since all of the states of $\mathcal{M}$ are reachable, each is equivalent to some state of $\mathcal{O}$. If $\mathcal{O}$ had fewer states than $\mathcal{M}$ we would have to have two states of $\mathcal{M}$ equivalent to the same state of $\mathcal{O}$, and so they would be equivalent to each other. This is impossible; we construct $\mathcal{M}$ to group together all equivalent states so two different states of $\mathcal{M}$ must be distinguishable. This means that $\mathcal{O}$ can't have fewer states than $\mathcal{M}$, and $\mathcal{M}$ really is minimal.

Here is a last fun fact about regular languages. Start with a fixed alphabet Σ and a language $\mathcal{L}$ over Σ. We say that strings x and y (not necessarily in $\mathcal{L}$) are $\mathcal{L}$-equivalent if for every string z either xz and yz are both in $\mathcal{L}$ or are both not in $\mathcal{L}$. (If you find this confusing, think of $\mathcal{L}$-equivalent strings as strings that take an automaton for the language to the same state.)

We can then talk about $\mathcal{L}$-equivalence classes -- the sets of strings that are all mutually $\mathcal{L}$-equivalent.

For example, consider $\mathcal{L} = 0^*1^*$. This divides all strings of 0's and 1's into 3 classes:

a) All strings in 0^*
b) All strings in 0^*1^+
c) All other strings

For instance, 000 and 001 are in different classes since 000 can be followed by a 0 to get a string in $\mathcal{L}$ and 001 can't.

The Myhill-Nerode Theorem say that a language $\mathcal{L}$ is regular if and only if it divides Σ^* into a finite number of $\mathcal{L}$-equivalence classes.

The proof uses the equivalence classes as states to build a DFA for the language.