

Kleene's Theorem (Part 1): If  $R$  is a regular expression over the alphabet  $\Sigma$ , and  $L$  is the language in  $\Sigma^*$  corresponding to  $R$ , then there is a finite automaton  $M$  that recognizes  $L$ .

We have already shown that the set of regular languages are closed under union, concatenation and Kleene closure.

By definition, every regular expression can be presented using only the operations union, concatenation and Kleene closure.

**Slide Lecture 6 -144**

1.  $\emptyset$  is a regular expression corresponding to the empty language.
2.  $\lambda$  is a regular expression corresponding to the language  $\{\lambda\}$ .
3. For each  $a \in \Sigma$ ,  $a$  is a regular expression corresponding to the language  $\{a\}$ .

**Slide Lecture 6 -145**

4. For any regular expressions  $r$  and  $s$  over  $\Sigma$  corresponding to the languages  $L_r$  and  $L_s$ , each of the following is a regular expression over  $\Sigma$  corresponding to the language indicated.

$(rs)$  corresponds to  $L_r L_s$

$(r + s)$  corresponds to  $L_r \cup L_s$

$(r^*)$  corresponds to  $L_r^*$

5. Only strings that can be produced using rules 1 to 4 are regular expressions.

**Slide Lecture 6 -146**

By definition for every regular language  
there is a regular expression.

We already know how to construct machines for rules 1 to 4.

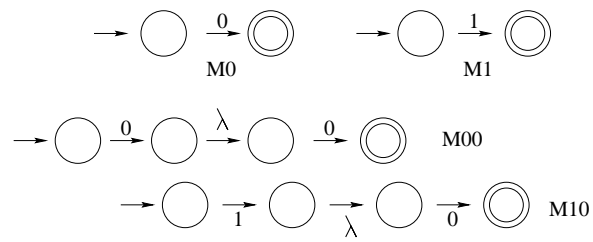
Since we have already shown that regular languages are closed under union, concatenation and Kleene Closure by showing that we can construction a corresponding N DFA- $\lambda$ , it follows that for every regular expression, there is an N DFA- $\lambda$  that accepts the corresponding regular language.

**Slide Lecture 6 -147**

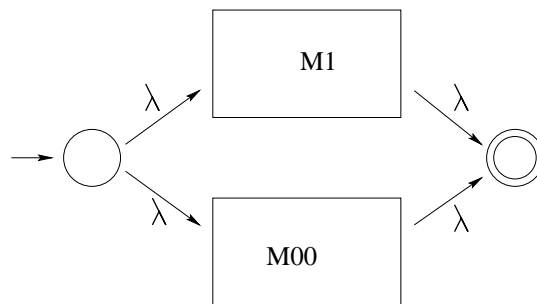
AN EXAMPLE:

Consider the regular expression  $(00 + 1)^*(10)^*$

Consider the following primitive machines.

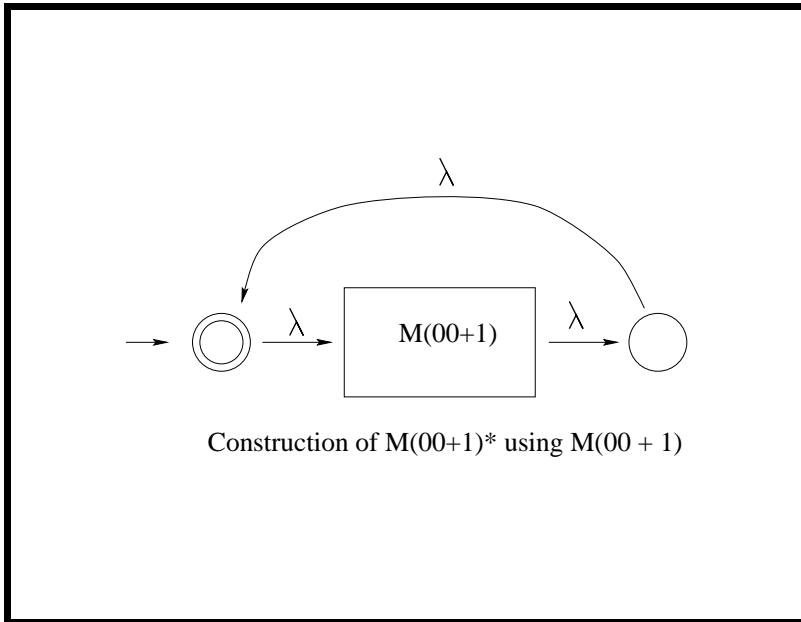


Slide Lecture 6 -148

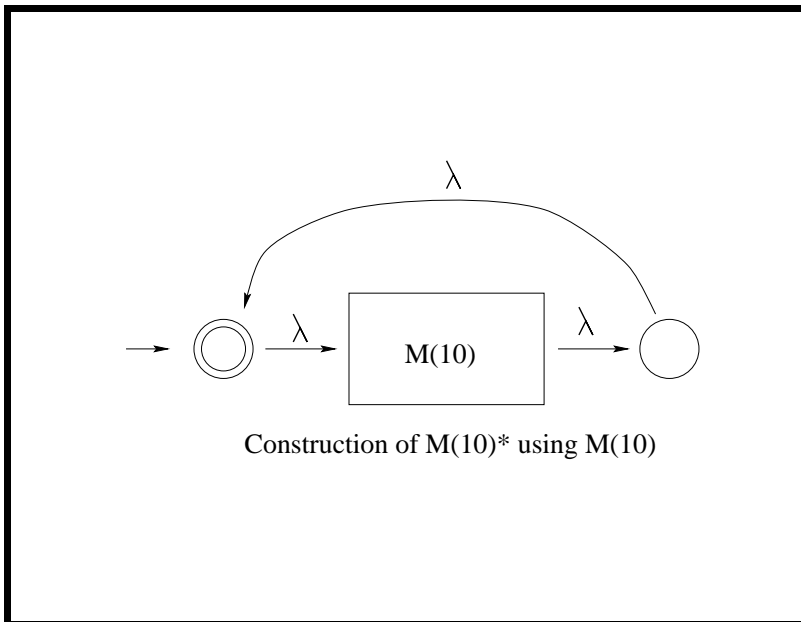


Construction of machine  $M(00+1)$  using  $M00$  and  $M1$ .

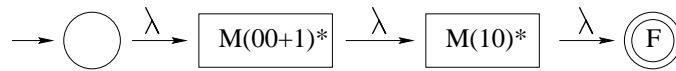
Slide Lecture 6 -149



Slide Lecture 6 -150



Slide Lecture 6 -151



Construction of  $M(00+1)^*(10)^*$

Slide Lecture 6 -152

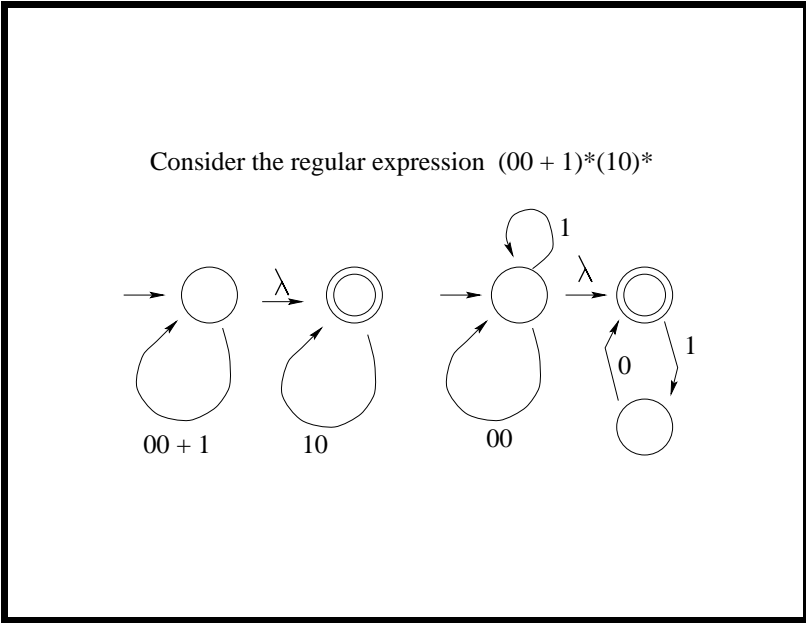
We can construct the machine much more directly if we wish, and in fact there are algorithms that directly construct machines for regular expression in compiler textbooks (e.g., Compilers by Aho, Sethi, Ullman 1988:121).

We can directly construction loops for Kleene closure.

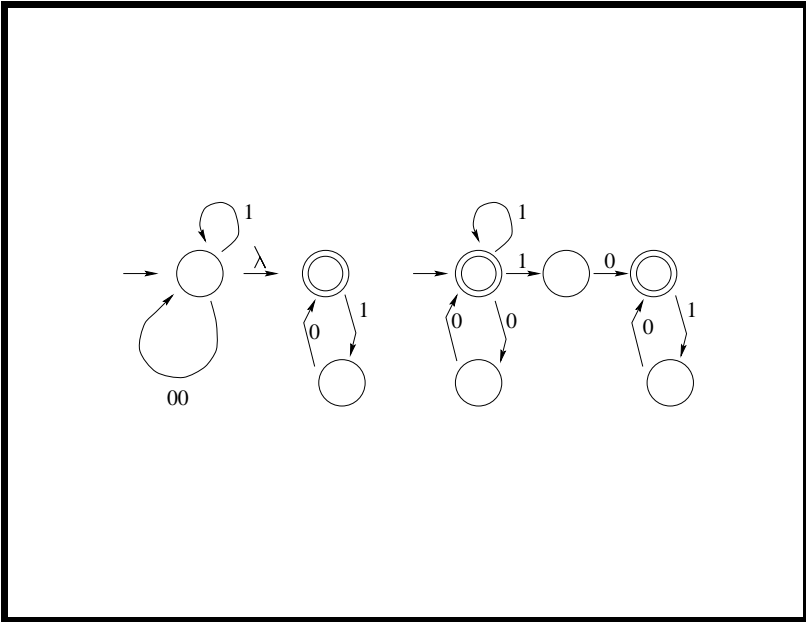
We can construction alternative paths for + operators.

Concatenation just generates sequences of transitions.

Slide Lecture 6 -153



Slide Lecture 6 -154



Slide Lecture 6 -155

Kleene's Theorem (Part 2): If  $M = (Q, \Sigma, \delta, S, F)$  is an FSA recognizing the language  $L$ , then there is a regular expression over the alphabet  $\Sigma$  corresponding to  $L$ .

Martin gives an inductive proof. We have shown that there is a construction algorithm using Set Equations.

**Slide Lecture 6 –156**

### **Minimal DFAs**

We now wish to know what is the minimal deterministic finite state machine that recognizes a particular language. Given any machine or regular expression for that language, we can construction the minimal DFA.

First, construct a DFA that recognizes the language. We can then apply a minimization algorithm.

**Slide Lecture 6 –157**

We are interested in states in the DFA that are equivalent:  
equivalent states compute the same thing.

Put another way, two states are equivalent if they  
“Right Invariant”.

Let us consider an informal definition of “right invariance.”

**Slide Lecture 6 –158**

Consider machine M.

Assume two states, A and B, are equivalent.

1. Construct 2 copies of machine M: machines M1 and M2.
2. Determine all the states that are “reachable” from state A in machine M1.
3. Determine all the states that are “reachable” from state B in machine M2.
4. Discard all unreachable states in the two machines.
5. Make states A and B the initial states of M1 and M2.

**Slide Lecture 6 –159**

If states A and B are right invariant, then  $L(M1) = L(M2)$  and A and B are equivalent states.

In machine M, there may be different computations (different string prefixes) that lead to state A or state B, but the computations (suffixes, hence the *right invariance*) processed after reaching state A or state B are identical.

**Slide Lecture 6 –160**

An alternative point of view is to talk about states that are distinguishable:

Two states A and B are *distinguishable* if they are not right invariant. If our reduced machines M1 and M2 with A and B as start states do not recognize the same language, then A and B are *distinguishable*.

**Slide Lecture 6 –161**

We can also talk about strings being distinguishable. If  $S$  is a set of strings, any two of which are distinguishable with respect to  $L$ ,  $S$  is a *pairwise distinguishable* (PD) set.

The most obvious way that strings can be *pairwise distinguishable* are if they lead to accepting final states or non-final rejecting states.

More generally, strings can also be distinguishable, and hence partitioned into equivalence classes, if they lead to nonequivalent states in a machine. So, strings are distinguishable if they lead to states that are distinguishable.

**Slide Lecture 6 –162**

Consider the relation  $I_L$  defined over  $\Sigma^*$ :

For  $x, y \in \Sigma^*$ ,  $xI_Ly$  means that  $x$  and  $y$  are indistinguishable with respect to  $L$ ; for every  $z \in \Sigma^*$ , either  $xz$  and  $yz$  are both in  $L$ , or  $xz$  and  $yz$  are both in the complement of  $L$ .

This means that prefixes  $x$  and  $y$  lead to states in a machine that recognizes  $L$  that are *right invariant*, and hence that these two states are equivalent.

But note that the definition is also independent of a machine. Not surprisingly, the relation  $I_L$  defines an equivalence relation.

**Slide Lecture 6 –163**

Lemma:  $I_L$  is an equivalence relation on  $\Sigma^*$ .

Proof:  $I_L$  is clearly reflexive and symmetric by definition. It is also transitive:

If  $xI_Ly$  and  $yI_Lw$  then  $xI_Lw$

Since  $xz \in L$  IFF  $yz \in L$  and  $yz \in L$  IFF  $wz \in L$  it follows that  $xI_Lw$ .

Put another way, prefix  $x$ ,  $y$  and  $w$  all lead to states that are right invariant, and therefore equivalent.

**Slide Lecture 6 -164**

Lemma:  $I_L$  is right invariant with respect to concatenation.

This means that if  $xI_Ly$ , then for all  $a \in \Sigma$ ,  $xaI_Lya$ . Similarly, if  $[x]$  is the equivalence class to which prefix  $x$  belongs, and  $[x] = [y]$ , then for all  $a \in \Sigma$ ,  $[xa] = [ya]$ .

Proof: Assume  $xI_Ly$  and let  $a \in \Sigma$ . We must show that for all  $z \in \Sigma$ ,  $xaz$  and  $yaz$  are either both in  $L$  or both not in  $L$ . By definition of  $I_L$ , we know that for every  $z' \in \Sigma^*$ , either  $xz'$  and  $yz'$  are both in  $L$ , or  $xz'$  and  $yz'$  are both in the complement of  $L$ . Choosing  $z' = az$  yields the desired result.

**Slide Lecture 6 -165**

Theorem: Let  $L$  be a regular language in  $\Sigma^*$ . Define  
 $M_L = (Q_L, \Sigma, \delta, S, F)$  where

1.  $Q_L$  is a finite set of equivalence classes with respect to  $I_L$ .
2.  $\Sigma$  is the input alphabet
3.  $\delta$  is the state transition function

$$\delta : Q \times \Sigma \longrightarrow Q$$

which is defined by  $\delta([x], a) = [xa]$  with respect to the equivalence class  $[x]$  for every  $x \in \Sigma^*$  and  $a \in \Sigma$ .

**Slide Lecture 6 –166**

4.  $S$  is the equivalence class (initial state) reachable by scanning  $\lambda$ :  $[\lambda]$
5.  $F$  is the set of final states,

$$\{q \in Q_L \mid q \cap L \neq \emptyset\}$$

$M_L$  is an FSA recognizing  $L$  and  $M_L$  has the fewest states of any FSA recognizing  $L$ .

Rather than formally deriving the proof, we will look at the corresponding algorithm for constructing a minimal machine.

**Slide Lecture 6 –167**

We are actually interested in finding states that are distinguishable.

We already know that strings that lead to accepting final states are distinguishable from strings that lead to non-final rejecting states.

So we know that final states and non-final states are distinguishable. (They are distinguishable with respect to  $\lambda$ .)

**Slide Lecture 6 –168**

Before motivating and presenting the main algorithm, we also note that we need only consider reachable states.

Occasionally, machines have states that are not reachable (e.g., by mistake).

This can't happen if one converts from an NFA to a DFA.

As a preprocessing step, remove unreachable states.

**Slide Lecture 6 –169**

Consider the following computation, for  $x \in \Sigma$ :

$$\delta(q_i, x) = q_j \in F$$

$$\delta(q_k, x) = q_l \notin F$$

We know in advance that  $q_j$  and  $q_l$  are distinguishable.

It follows that states  $q_i$  and  $q_k$  are distinguishable.

Recursively, for any pair of transitions of the form

$$\delta(q_i, x) = q_j \quad \delta(q_k, x) = q_l$$

states  $q_i$  and  $q_k$  are distinguish if,  
for any  $x \in \Sigma$ ,  $q_j$  and  $q_l$  are distinguish.

**Slide Lecture 6 –170**

So our algorithm works as follows over all pairs of states.

1. Mark final states and non-final states as distinguishable.

2. Recursively iterating over all pairs of states.

For any transition of the form

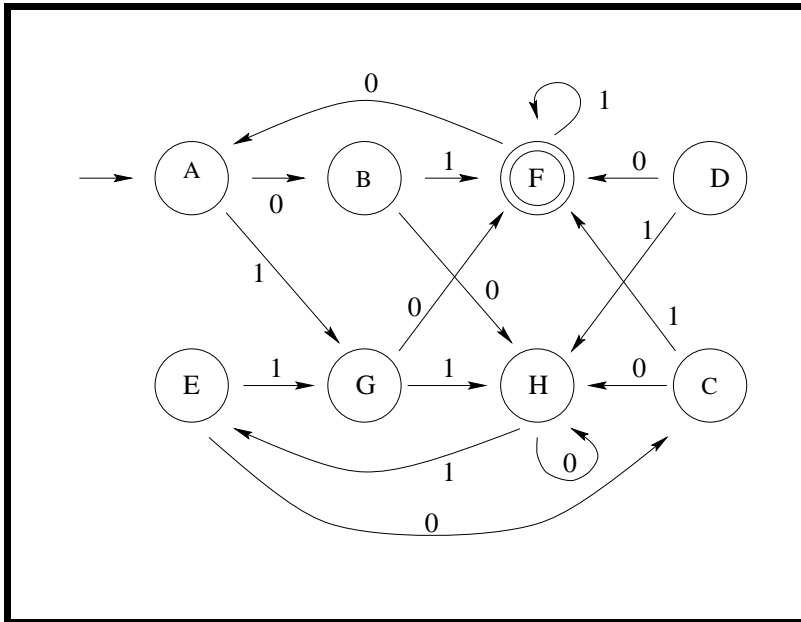
$$\delta(q_i, x) = q_j \quad \delta(q_k, x) = q_l$$

and for some  $x \in \Sigma$ , if  $q_j$  and  $q_l$  are distinguishable, mark  $q_i$  and  $q_k$  as distinguish.

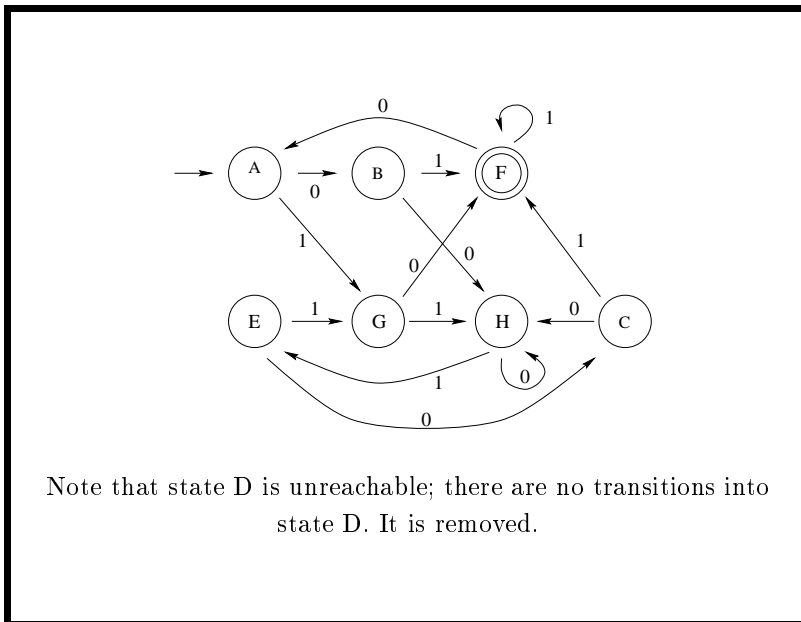
3. If on any iteration over all possible state pairs one fails to find a new pair of states that are distinguishable, terminate.

4. All states that are not distinguishable are equivalent.

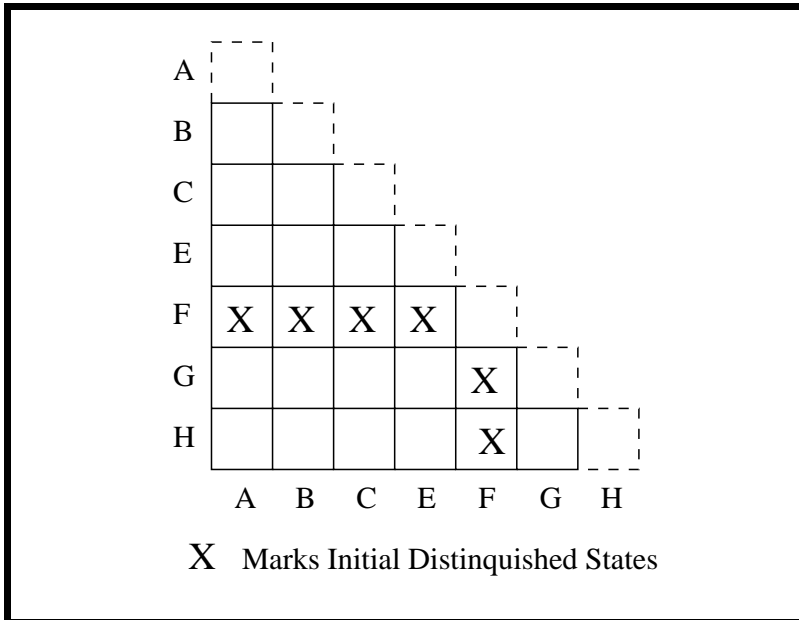
**Slide Lecture 6 –171**



Slide Lecture 6 -172



Slide Lecture 6 -173



Slide Lecture 6 -174

Consider pair (A,B)

$$\delta(A,0) = B \quad \delta(B,0) = H$$

$$\delta(A,1) = G \quad \delta(B,1) = F$$

$A \equiv B$  IFF  $H \equiv B$  and  $G \equiv F$ .

Since G is distinguishable from F, A is distinguishable from B.

Slide Lecture 6 -175

Consider pair (A,C)

$$\delta(A,0) = B \quad \delta(C,0) = H$$

$$\delta(A,1) = G \quad \delta(C,1) = F$$

Since G is distinguishable from F, A is distinguishable from C.

\* \* \*

Consider pair (A,E)

$$\delta(A,0) = B \quad \delta(E,0) = C$$

$$\delta(A,1) = G \quad \delta(E,1) = G$$

Since both A and E go to G on a 1,  $A \equiv E$  IFF  $B \equiv C$ .

**Slide Lecture 6 -176**

Consider pair (A,G)

$$\delta(A,0) = B \quad \delta(G,0) = F$$

$$\delta(A,1) = G \quad \delta(G,1) = H$$

Since B is distinguishable from F, A is distinguishable from G.

\* \* \*

Consider pair (A,H)

$$\delta(A,0) = B \quad \delta(H,0) = H$$

$$\delta(A,1) = G \quad \delta(H,1) = E$$

Our algorithm can't say if A is distinguishable from H yet.

But note if  $A \equiv H$  then  $B \equiv H$  and  $A \equiv B$ .

**Slide Lecture 6 -177**

Consider pair (B,C)

$$\delta(B,0) = H \quad \delta(C,0) = H$$

$$\delta(B,1) = F \quad \delta(C,1) = F$$

States B and C are equivalent, since they are clearly right invariant.  $B \equiv C$  IFF  $H \equiv H$  and  $F \equiv F$ .

\* \* \*

Consider pair (B,E)

$$\delta(B,0) = H \quad \delta(E,0) = C$$

$$\delta(B,1) = F \quad \delta(E,1) = G$$

$B \equiv E$  IFF  $H \equiv C$  and  $F \equiv F$ . Since G is distinguishable from F, B is distinguishable from E.

**Slide Lecture 6 -178**

Consider pair (B,G)

$$\delta(B,0) = H \quad \delta(G,0) = F$$

$$\delta(B,1) = F \quad \delta(G,1) = H$$

Since H is distinguishable from F, B is distinguishable from G.

\* \* \*

Consider pair (B,H)

$$\delta(B,0) = H \quad \delta(H,0) = H$$

$$\delta(B,1) = F \quad \delta(H,1) = E$$

Since E is distinguishable from F, B is distinguishable from H.

**Slide Lecture 6 -179**

Since we know C has exactly the same computation as B, C is also distinguishable from the same states as B.

$$(B \neq E) \longrightarrow (C \neq E)$$

$$(B \neq G) \longrightarrow (C \neq G)$$

$$(B \neq H) \longrightarrow (C \neq H)$$

**Slide Lecture 6 -180**

Consider pair (E,G)

$$\delta(E, 0) = C \quad \delta(G, 0) = F$$

$$\delta(E, 1) = G \quad \delta(G, 1) = H$$

Since C is distinguishable from F,  $E \neq G$ .

\* \* \*

Consider pair (E,H)

$$\delta(E, 0) = C \quad \delta(H, 0) = H$$

$$\delta(E, 1) = G \quad \delta(H, 1) = E$$

$E \equiv H$  IFF  $C \equiv H$  and  $G \equiv E$ .

Mark E and H as distinguishable.

Note E and H are distinguishable with respect to information generated on this same pass.

**Slide Lecture 6 -181**

Consider pair (G,H)

$$\delta(G,0) = F \quad \delta(H,0) = H$$

$$\delta(G,1) = H \quad \delta(H,1) = E$$

$G \equiv H$  IFF  $F \equiv H$  and  $H \equiv E$ .  
Mark G and H as distinguishable since  $F \neq H$ .

**Slide Lecture 6 -182**

This completes one iteration over the table.

The only unresolved states are (A,E) and (A,H).

$$\delta(A,0) = B \quad \delta(E,0) = C$$

$$\delta(A,1) = G \quad \delta(E,1) = G$$

Since we know that  $B \equiv C$  that implies  $A \equiv E$ .

$$\delta(A,0) = B \quad \delta(H,0) = H$$

$$\delta(A,1) = G \quad \delta(H,1) = E$$

Since we now know that  $B \neq H$  this implies  $A \neq H$ .

**Slide Lecture 6 -183**

|   |       |       |       |       |       |       |   |
|---|-------|-------|-------|-------|-------|-------|---|
| A |       |       |       |       |       |       |   |
| B | $X_1$ |       |       |       |       |       |   |
| C | $X_1$ | S     |       |       |       |       |   |
| E | S     | $X_1$ | $X_1$ |       |       |       |   |
| F | $X_0$ | $X_0$ | $X_0$ | $X_0$ |       |       |   |
| G | $X_1$ | $X_1$ | $X_1$ | $X_1$ | $X_0$ |       |   |
| H | $X_2$ | $X_1$ | $X_1$ | $X_2$ | $X_0$ | $X_1$ |   |
|   | A     | B     | C     | E     | F     | G     | H |

$X_t$  Marks Initial Distinguished States  
 $t$  Marks the iteration