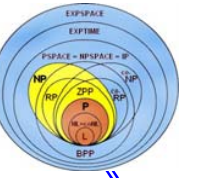
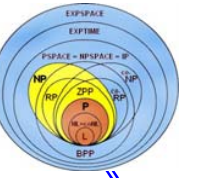
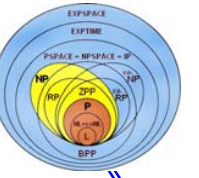


Session 6

- Decision Properties of Regular Languages
- Equivalence of Regular Languages
- Minimization of Deterministic Finite Automata

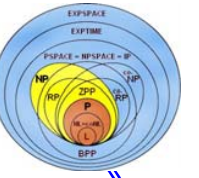


Decision Properties of Regular Languages



The decision problem for regular languages is a question that takes any regular language (often automaton) as input and asks for a terminating algorithm yielding a boolean answer.

Such problem is typically a question about the set of strings; answers that involve running the machine on every string in the set are not useful, since they will take forever. That is not allowed: in every case, a decision algorithm must return a correct answer in *finite* time.

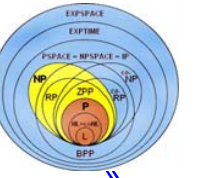


Here is the list of the decision problems for regular languages:

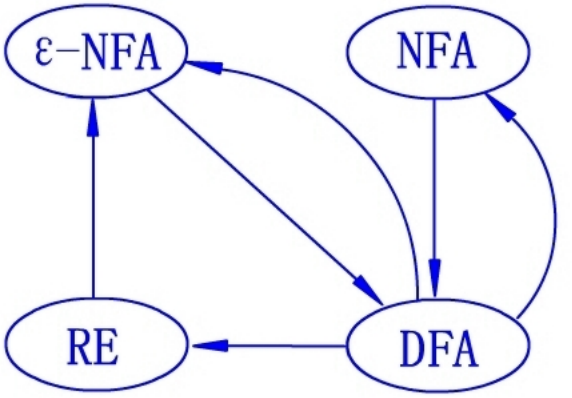
- Given a string w and a regular language L, is $w \in L$?
- Given a regular language L, is $L = \emptyset$? $L = \Sigma^*$? L finite?
- Given two regular languages L_1 and L_2, is $L_1 \cap L_2 = \emptyset$? $L_1 \subseteq L_2$? $L_1 = L_2$?

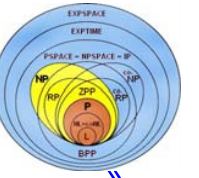
These problems *do* have algorithm that correctly answer the question. Some of the algorithm differ in how *efficient* they are.

For some decision problems, as we shall see later in the course, that answer is, surprisingly, no.



The regular languages can be represented by DFA, NFA, ϵ-NFA or regular expression. We know that we can convert any of the four representations to any of the other three representations. We shall, first, consider the time complexity of each of the conversions.



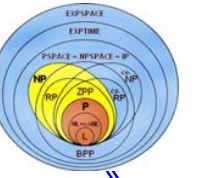


From NFA to DFA

Suppose the ϵ-NFA has n states. To compute ECLOSE(p) we follow at most n^2 arcs.

The DFA has 2^n states, for each state S and each $a \in \Sigma$ we compute $\delta_D(S, a)$ in n^3 steps. Grand total is $O(n^3 2^n)$ steps.

If we compute δ for reachable states only, we need to computer $\delta_D(S, a)$ only s times, where s in the number of reachable states. Grand total is $O(n^3 s)$ steps.



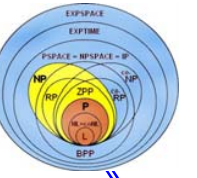
From DFA to NFA

All we need to do is to put set brackets around the states. Total $O(n)$ steps.

From FA to regular expression

We need to compute n^3 entries of size up to 4^n. Total is $O(n^3 4^n)$.

The FA is allowed to be a NFA. If we first wanted to convert the NFA to a DFA, the total time would be doubly exponential: $O(n^3 4^{n^3 2^n})$.

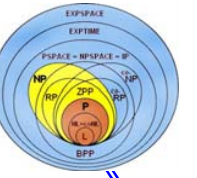


From regular expression to FA's

We can build an expression tree for the regular expression in n steps, and work up the tree, constructing the ϵ-NFA for each node. The numbers of states and arcs of the resulting ϵ-NFA are both $O(n)$.

We can eliminate ϵ-transitions from an n-state ϵ-NFA, to make an ordinary NFA, in $O(n^3)$ time.

If we want a DFA, we have to need an exponential number of steps.

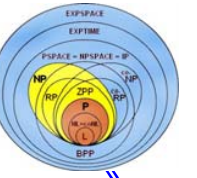


Testing Membership in a Regular Language

Given a string w and a regular language L, test $w \in L$.

If L is represented by a DFA, the algorithm is simple. Simulate the DFA processing the string of input symbols w, beginning in the start state. If the DFA ends in an accepting state, the answer is "yes"; otherwise the answer is "no".

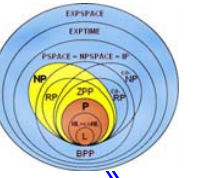
If $|w| = n$, this takes $O(n)$ steps, extremely fast.



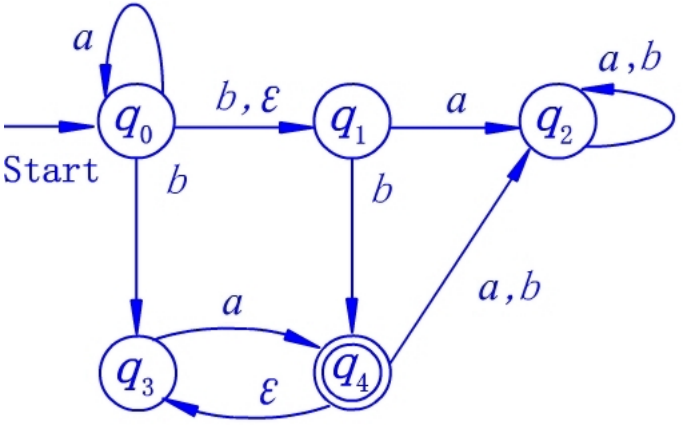
If L is represented by an NFA, we can translate the NFA to equivalent FA by subset construction and then run the DFA on the string. However, we would like to avoid mapping from NFA's to DFA's, since simulating the NFA directly is more efficient.

If $|w| = n$, and NFA has s states, it takes $O(ns^2)$ steps.

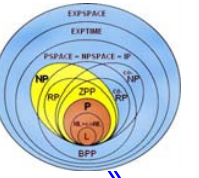
If the NFA has ϵ-transitions, then we must compute the ϵ-closure before starting the simulation. In this case, simulating NFA on w takes $O(ns^3)$.



Example Does the following ϵ-NFA accept the string *aaaba*?

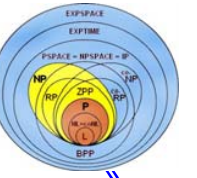


The initial set of states that the machine could be in is $\{q_0, q_1\}$. We then have the following table, showing how the set of possible current states changes with each new transition.



input symbol	possible current states
	$\{q_0, q_1\}$
a $\downarrow$	
	$\{q_0, q_1, q_2\}$
a $\downarrow$	
	$\{q_0, q_1, q_2\}$
a $\downarrow$	
	$\{q_0, q_1, q_2\}$
b $\downarrow$	
	$\{q_1, q_2, q_3, q_4\}$
a $\downarrow$	
	$\{q_2, q_3, q_4\}$

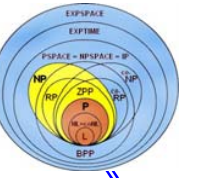
After the string has been processed, we examine the set of possible states $\{q_2, q_3, q_4\}$ and find $\{q_4\}$, so the answer returned is "yes".



In an implementation, the set of possible current states would be kept in a data structure, and each transition would be cause states to be added or deleted from the set.

Once the string was fully processed, all that needs to be done is to take intersection between the accept states and the set of possible current states.

If L is represented by a regular expression of length s, we first convert to an ϵ-NFA with $2s$ states. Then we simulate w on this machine, in $O(ns^3)$ steps.



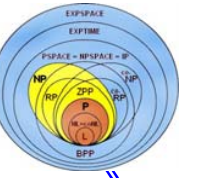
Testing Emptiness in a Regular Language

If the given language L is represented by an FA, the emptiness question is whether there is any path from the start state to some accepting sate. Deciding whether we can reach an accepting state from the start is a simple instance of graph-reachability.

The algorithm can be summarized by a recursive process.

Basis step: The start state q_0 is surely reachable.

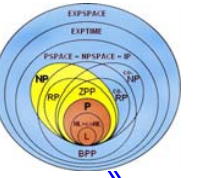
Inductive step: If state q is reachable, and there is an arc from q to p with any label, then p is reachable.



We compute the set of reachable states in this manner. If any accepting sate is among them, we answer "no", and otherwise we answer "yes".

The reachability calculation takes no more time than $O(n^2)$ if the automaton has n states, and in fact it is no worse than proportional to the number of arcs in the automaton's transition diagram, which could be less than n^2.

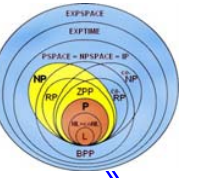
If we are given a regular expression representing the language L, we could convert the expression to an ϵ-NFA and proceed as above.



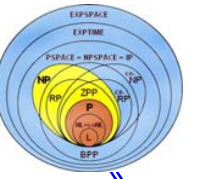
Use the emptiness checker and closure properties for regular language, we can give algorithms for some other decision problems such as is $L = \Sigma^*$? $L_1 \cap L_2 = \emptyset$? $L_1 \subseteq L_2$? $L_1 = L_2$?

Example Given DFA's A_1 and A_2, we wish to see if there is some string that both machine accept. The following algorithm performs this task:

- Build the product machine $A_1 \times A_2$, such that $L(A_1 \times A_2) = L(A_1) \cap L(A_2)$. This machine only accepts strings accepted by both A_1 and A_2.
- Run the emptiness checker on $A_1 \times A_2$, and return its answer.



Equivalence of Regular Languages



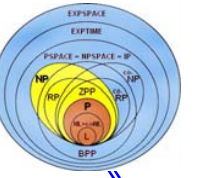
Testing Equivalence of States

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and $\{p, q\} \subseteq Q$. We define

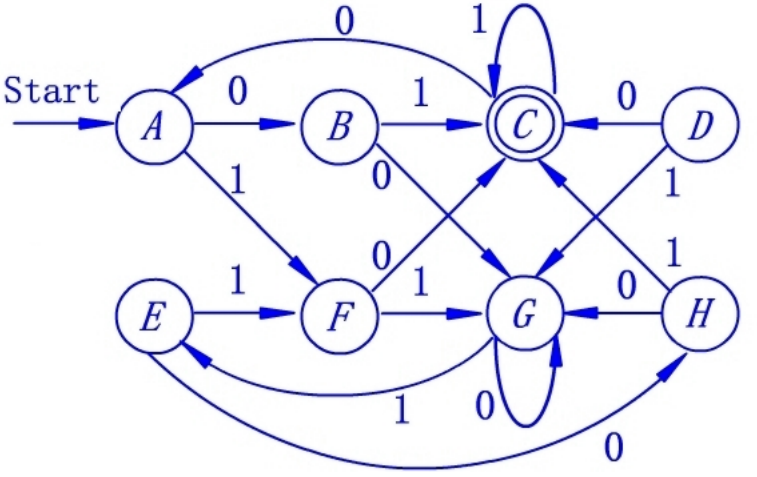
$$p \equiv q \;\Leftrightarrow\; \forall\, w \in \Sigma^* :\; \hat{\delta}(p, w) \in F \text{ iff } \hat{\delta}(q, w) \in F$$

If $p \equiv q$ we say that p and q are equivalent. Otherwise, we say that p and q are distinguishable. In other words, $p \not\equiv q$ if and only if

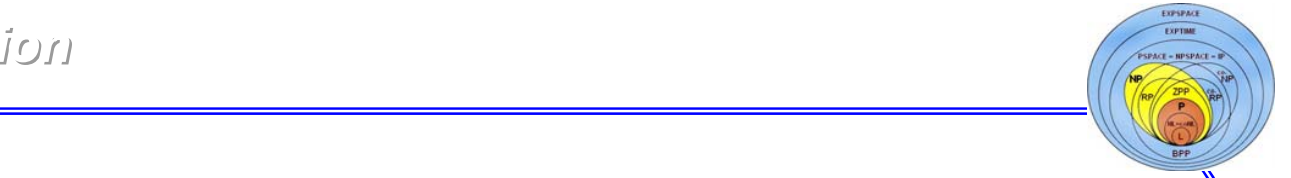
$$\exists\, w :\; \hat{\delta}(p, w) \in F \text{ and } \hat{\delta}(q, w) \notin F, \text{ or vice verse}$$



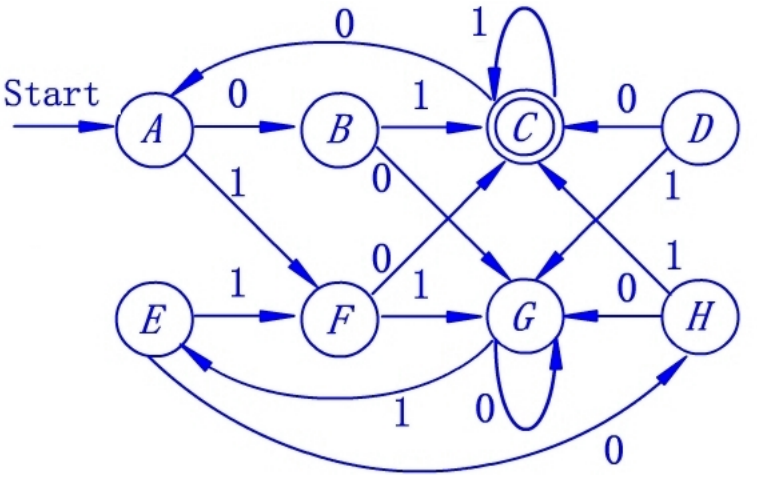
Example Consider DFA $A = (Q, \Sigma, \delta, q_0, F)$.



Note that $\hat{\delta}(C, \epsilon) \in F,\ \hat{\delta}(G, \epsilon) \notin F \ \Rightarrow C \not\equiv G$



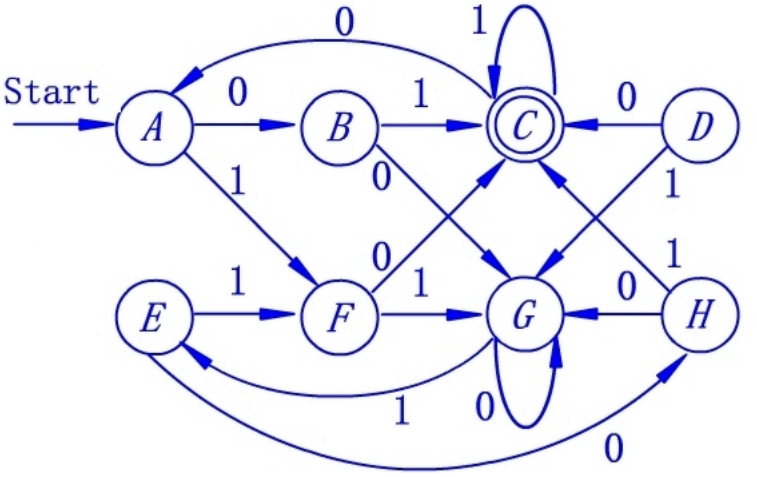
Also $\hat{\delta}(A, 01) = C \in F,\ \hat{\delta}(G, 01) = E \notin F \ \Rightarrow A \not\equiv G$



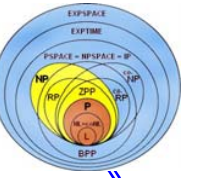
What about A and E?

Since $\hat{\delta}(A,\epsilon) = A \notin F,\ \hat{\delta}(E,\epsilon) = E \notin F, \hat{\delta}(A,1) = F = \hat{\delta}(E,1).$

Therefore $\hat{\delta}(A,1x) = \hat{\delta}(E,1x) = \hat{\delta}(F,x).$



And $\hat{\delta}(A,00) = G = \hat{\delta}(E,00),\ \hat{\delta}(A,01) = C = \hat{\delta}(E,01).$ Conclusion: $A \equiv E.$

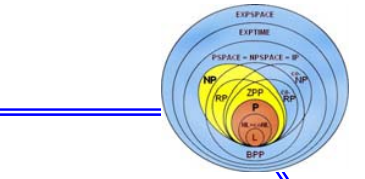


We can compute distinguishable pairs in a DFA $A = (Q, \Sigma, \delta, q_0, F)$ with the following inductive table-filling algorithm (TF-algorithm):

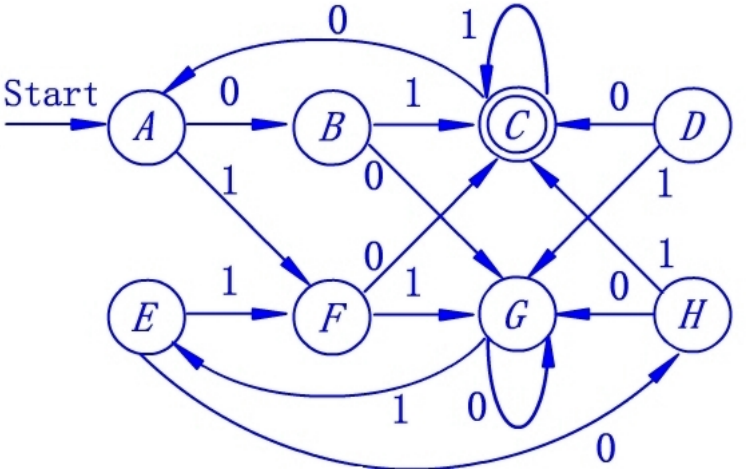
Basis step: If $p \in F$ and $q \notin F$, then $p \not\equiv q$.

Inductive step: If $\exists a \in \Sigma$ such that $p = \delta(r, a) \not\equiv q = \delta(s, a)$, then $r \not\equiv s$.

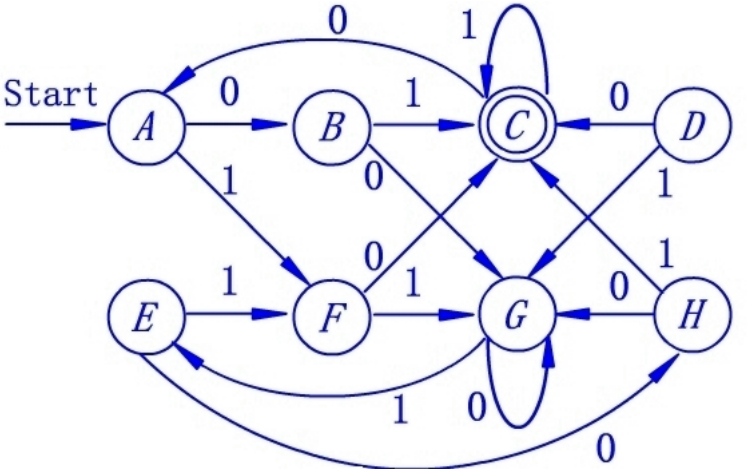
The reason of the induction rule is that there must be some string w such that exactly one of $\hat{\delta}(p, w)$ and $\hat{\delta}(q, w)$ is accepting. Then string aw must distinguish r from s.



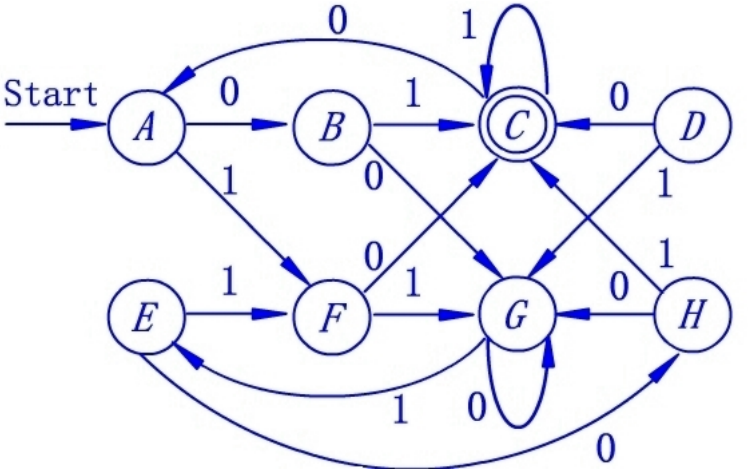
Example Apply the TF-algorithm to A



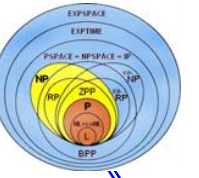
	A	B	C	D	E	F	G
B							
C	x	x					
D			x				
E			x				
F			x				
G			x				
H			x				



	A	B	C	D	E	F	G
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G		x	x	x		x	
H	x		x	x	x	x	x



	A	B	C	D	E	F	G
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x

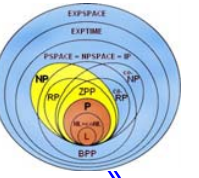


Theorem 4.8 *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and $\{p, q\} \subseteq Q$. If p and q are not distinguishable by TF-algorithm, then $p \equiv q$.*

Proof Suppose to the contrary that there is a *bad pair* $\{p, q\}$, such that

- $\exists w: \ \hat{\delta}(p, w) \in F$ and $\hat{\delta}(q, w) \notin F$, or vice verse,
- The Tf-algorithm does not distinguish between p and q.

If there are bad pairs, then there must be some that are distinguished by the shortest strings among all those strings that distinguish bad pairs.

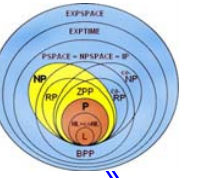


Let $\{p, q\}$ be one such bad pair, and let $w = a_1 a_2 \cdots a_n$ be the shortest string that identified a bad pair $\{p, q\}$.

Now $w \neq \epsilon$ since otherwise the TF-algorithm would in the basis step distinguish p from q. Thus $n \geq 1$.

Consider states $r = \delta(p, a_1)$ and $s = \delta(q, a_1)$. Now $\{r, s\}$ cannot be a bad pair since $\{r, s\}$ would be identified by a string shorter than w. Therefore, the TF-algorithm must have discovered that r and s are distinguishable.

But then the TF-algorithm would distinguish p from q in the inductive step. Thus there are no bad pair and the theorem is true. ◀

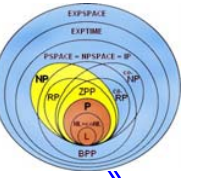


Testing Equivalence of Regular Languages

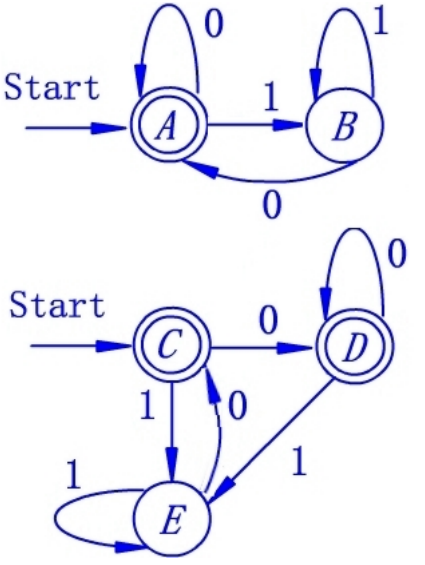
The TF-algorithm gives us an easy way to test if two languages are the same.

Let L and M be regular languages, each given in some form. To test if $L = M$?

- Convert both L and M to DFA's.
- Imagine the DFA that is the union of the two DFA's (never mind there are two start states).
- If TF-algorithm says that the two start states are distinguishable, then $L \neq M$, otherwise $L = M$.

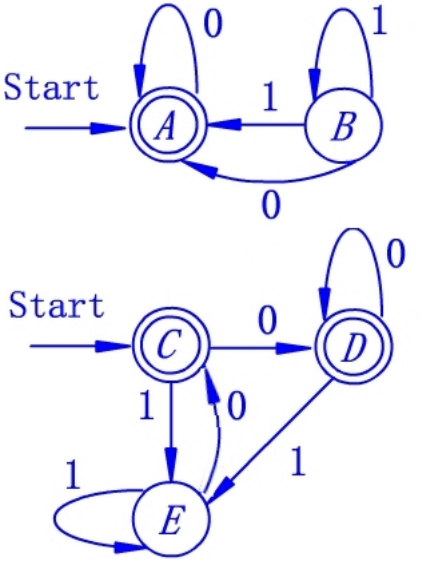


Example Consider the two DFA's.



We can see that both DFA accept the language $L(\epsilon + (\mathbf{0} + \mathbf{1})^*\mathbf{0})$.

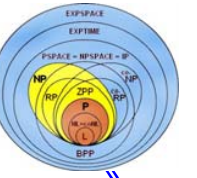
The result of the TF-algorithm is



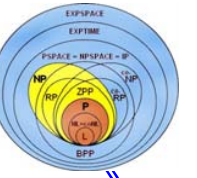
	A	B	C	D
B	x			
C		x		
D		x		
E	x		x	x

Since A and C are found equivalent. Therefore the two automata are equivalent.


Break for 15 minutes

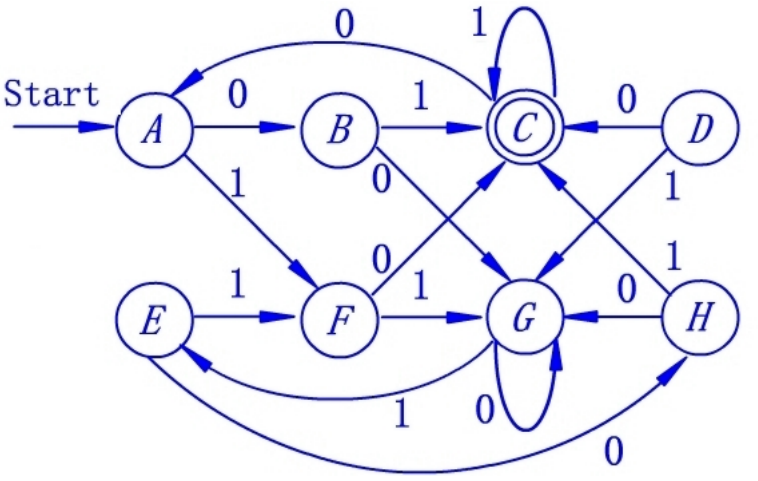


Minimization of Deterministic Finite Automata

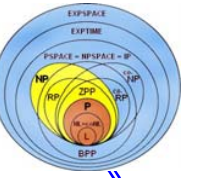


We can use the TF-algorithm to minimize a DFA by merging all equivalent states. In other words, replace each state p by $p/_{\equiv}$. This minimum-state DFA is unique for the language.

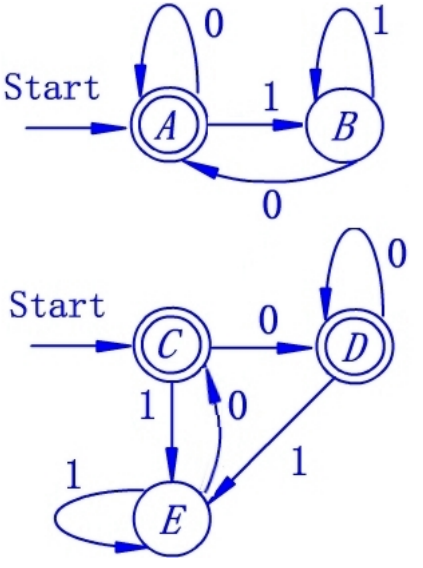
The idea behind the algorithm is to partition the states (eliminating any state that can no be reached from the start state) into blocks, so that all states in the same block are equivalent, and no pair of states from different blocks are equivalent.

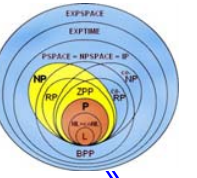


Example The DFA A has equivalence classes $\{\{A, E\}, \{B, H\}, \{C\}, \{D, F\}, \{G\}\}$.



Example The following DFA has equivalence classes $\{\{A, C, D\}, \{B, E\}\}$.



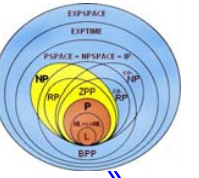


Theorem 4.9 *The equivalence of states is transitive. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and $\{p, q, r\} \subseteq Q$. If $p \equiv q$ and $q \equiv r$, then $p \equiv r$.*

Proof Suppose to the contrary that $p \not\equiv r$, then $\exists w$ such that $\hat{\delta}(p, w) \in F$ and $\hat{\delta}(r, w) \notin F$, or vice verse.

On the other hand, $\hat{\delta}(q, w)$ is either accepting state or not. If $\hat{\delta}(q, w) \in F$, then $q \not\equiv r$; If $\hat{\delta}(q, w) \notin F$, then $q \not\equiv p$. The vice verse case is proved symmetrically.

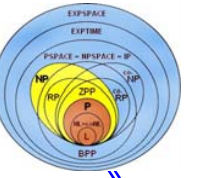
Thus it must be that $p \equiv r$. ◄



We can use Theorem 4.9 to justify the obvious algorithm for partitioning states. For each state q, construct a block that consists of q and all the states that are equivalent to q.

We must show that the resulting blocks are a partition; i.e. no state is in two distinct blocks.

Theorem 4.10 *If we create for each state q of a DFA a block consisting of q and all the states that are equivalent to q, then the different blocks of states form a partition of the set of states.*

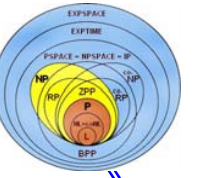


Now, we state the algorithm for minimizing a DFA $A = (Q, \Sigma, \delta, q_0, F)$.

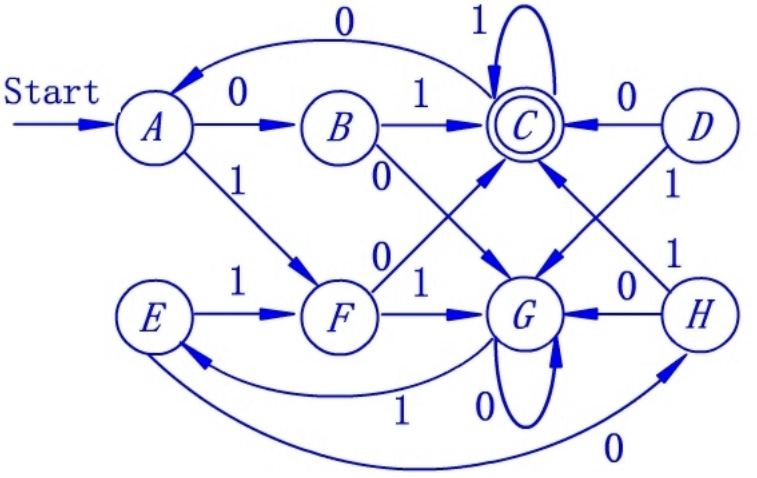
1. Use the TF-algorithm to find all the pairs of equivalent states.
2. Partition the set of states Q into blocks of mutually equivalent states.
3. Construct the minimum-state equivalent DFA $B = (Q/_{\equiv}, \Sigma, \gamma, q_0/_{\equiv}, F/_{\equiv})$, where

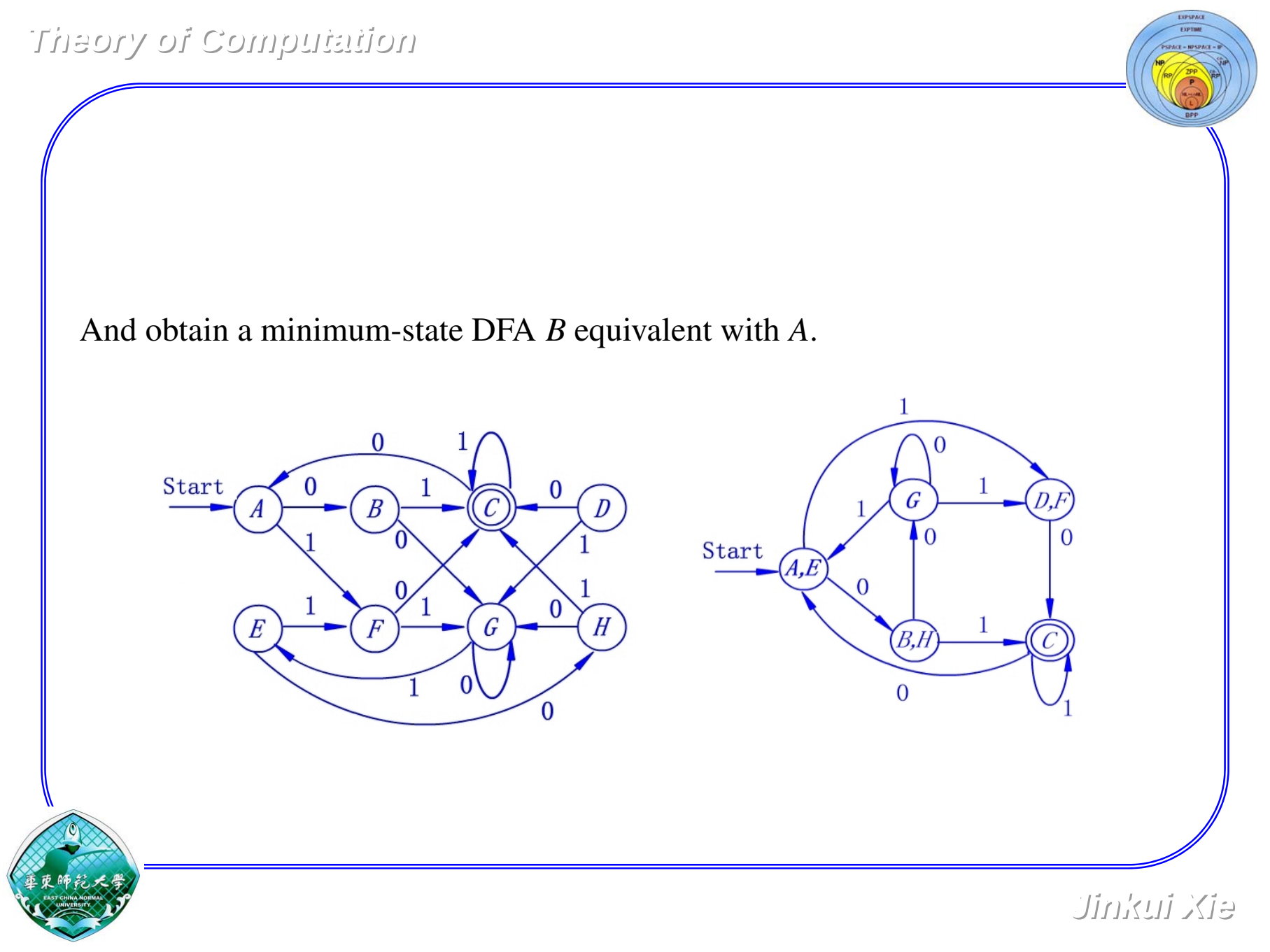
$$\gamma(q/_{\equiv}, a) = \delta(q, a)/_{\equiv}.$$

In order for B to be well defined we have to show that "if $p \equiv q$ then $\delta(p, a) \equiv \delta(q, a)$". By the TF-algorithm, we know it is true. Note also that $F/_{\equiv}$ contains all and only the accepting states of A.

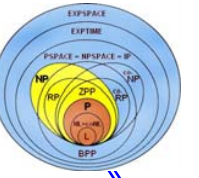


Example We can minimize DFA A.





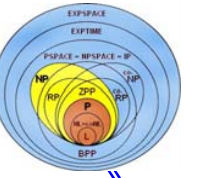
And obtain a minimum-state DFA B equivalent with A.



Theorem 4.11 *Given any DFA $A = (Q, \Sigma, \delta, q_0, F)$, application of the minimize algorithm yields another DFA $B = (Q/_{\equiv}, \Sigma, \gamma, q_0/_{\equiv}, F/_{\equiv})$ such that $L(A) = L(B)$. Furthermore, B is minimal in the sense that there is no other DFA with a smaller number of states which also accepts $L(A)$.*

Proof There are two steps. The first is to show DFA B equivalent with A. This is relatively easy and we can use inductive arguments to prove

$$\hat{\delta}(q_0, w)/_{\equiv} = \hat{\gamma}(q_0/_{\equiv}, w).$$

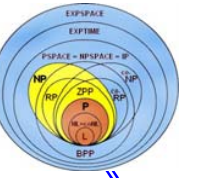


The second part, to show that B is minimal, is harder.

Suppose B has states $\{p_0, p_1, \cdots, p_m\}$, with p_0 the start state. Assume that there is an equivalent DFA C, with transition function η and start state r_0, equivalent to B, but with fewer states.

Since there are no inaccessible states in B, there must be distinct strings $w_1, w_2, \cdots, w_m$ such that

$$\hat{\gamma}(p_0, w_i) = p_i, \quad i = 1, 2, \cdots, m.$$

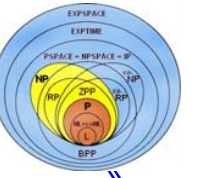


But since C has fewer states than B, there must be at least two of these strings, say w_k and w_l, such that

$$\hat{\eta}(r_0, w_k) = \hat{\eta}(r_0, w_l).$$

By the construction of B, p_k and p_l are distinguishable, there must be some string x such that $\hat{\gamma}(p_0, w_k x) = \hat{\gamma}(p_k, x)$ is a final state, and $\hat{\gamma}(p_0, w_l x) = \hat{\gamma}(p_l, x)$ is a nonfinal state (or vice verse).

In other words, $w_k x$ is accepted by B and $w_l x$ is not.

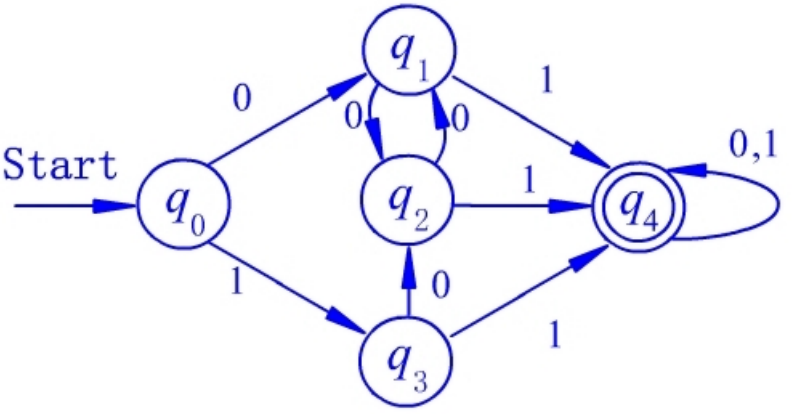


But note that

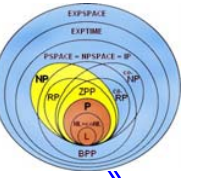
$$\hat{\eta}(r_0, w_k x) = \hat{\eta}(\hat{\eta}(r_0, w_k), x) = \hat{\eta}(\hat{\eta}(r_0, w_l), x) = \hat{\eta}(r_0, w_l x).$$

Thus, C either accepts both $w_k x$ and $w_l x$ or rejects both, contradicting the assumption that B and C are equivalent.

This contradiction proves that C cannot exists. ◄



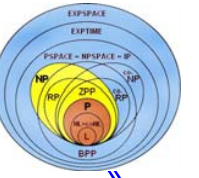
Example Minimizing the following DFA A.



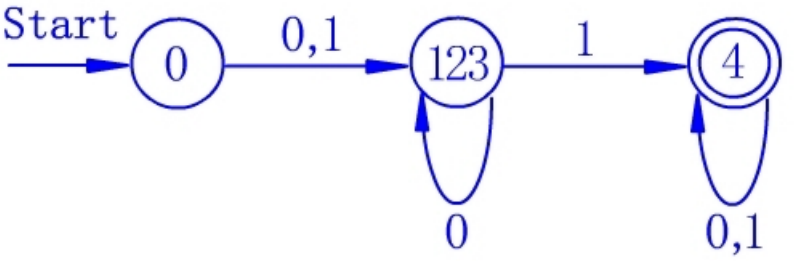
Solution The basis step in TF-algorithm will identify (mark) distinguishable pairs $\{q_0, q_4\}$, $\{q_1, q_4\}$, $\{q_2, q_4\}$ and $\{q_3, q_4\}$.

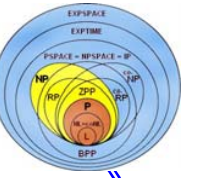
In some pass through the inductive step loop, the TF-algorithm computes $\delta(q_1, 1) = q_4$ and $\delta(q_0, 1) = q_3$. Since $\{q_3, q_4\}$ is a distinguishable pair, the pair $\{q_0, q_1\}$ is also marked.

Continuing this way, the TF-algorithm eventually marks the pairs $\{q_0, q_1\}$, $\{q_0, q_2\}$, $\{q_0, q_3\}$, $\{q_0, q_4\}$, $\{q_1, q_4\}$, $\{q_2, q_4\}$ and $\{q_3, q_4\}$ as distinguishable, leaving the indistinguishable pairs $\{q_1, q_2\}$, $\{q_1, q_3\}$ and $\{q_2, q_3\}$.

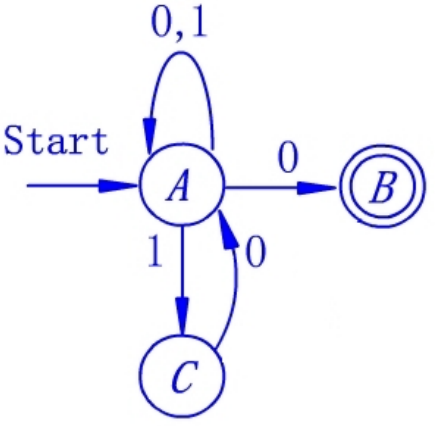


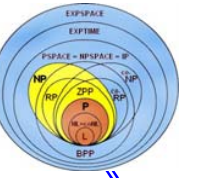
Therefore, the states q_1, q_2, q_3 are all equivalent, and all of the states have been partitioned into the sets $\{q_0\}$, $\{q_1, q_2, q_3\}$ and $\{q_4\}$. Applying minimize algorithm yields the DFA B follows.





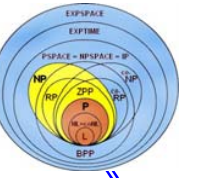
☞ Note that we cannot apply the TF-algorithm to NFA's. For example. to minimize following NFA we simply remove state C. However, $A \not\equiv C$.




EXPSPACE
EXPTIME
PSPACE = NPSPACE = IP
NP
RP
ZPP
P
NP
RP
L
BPP

Thank you!


華東師範大學
EAST CHINA NORMAL UNIVERSITY


EXPSPACE
EXPTIME
PSPACE = NPSPACE = IP
NP
RP
ZPP
P
L
BPP


華東師範大學
EAST CHINA NORMAL UNIVERSITY