Introduction to the Theory of Computation

XU Ming

School of Software Engineering, East China Normal University

April 24, 2024

OUTLINE

- Equivalence of Automata
- Minimization of Automata

Equivalence of Automata

Testing Equivalence of States

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

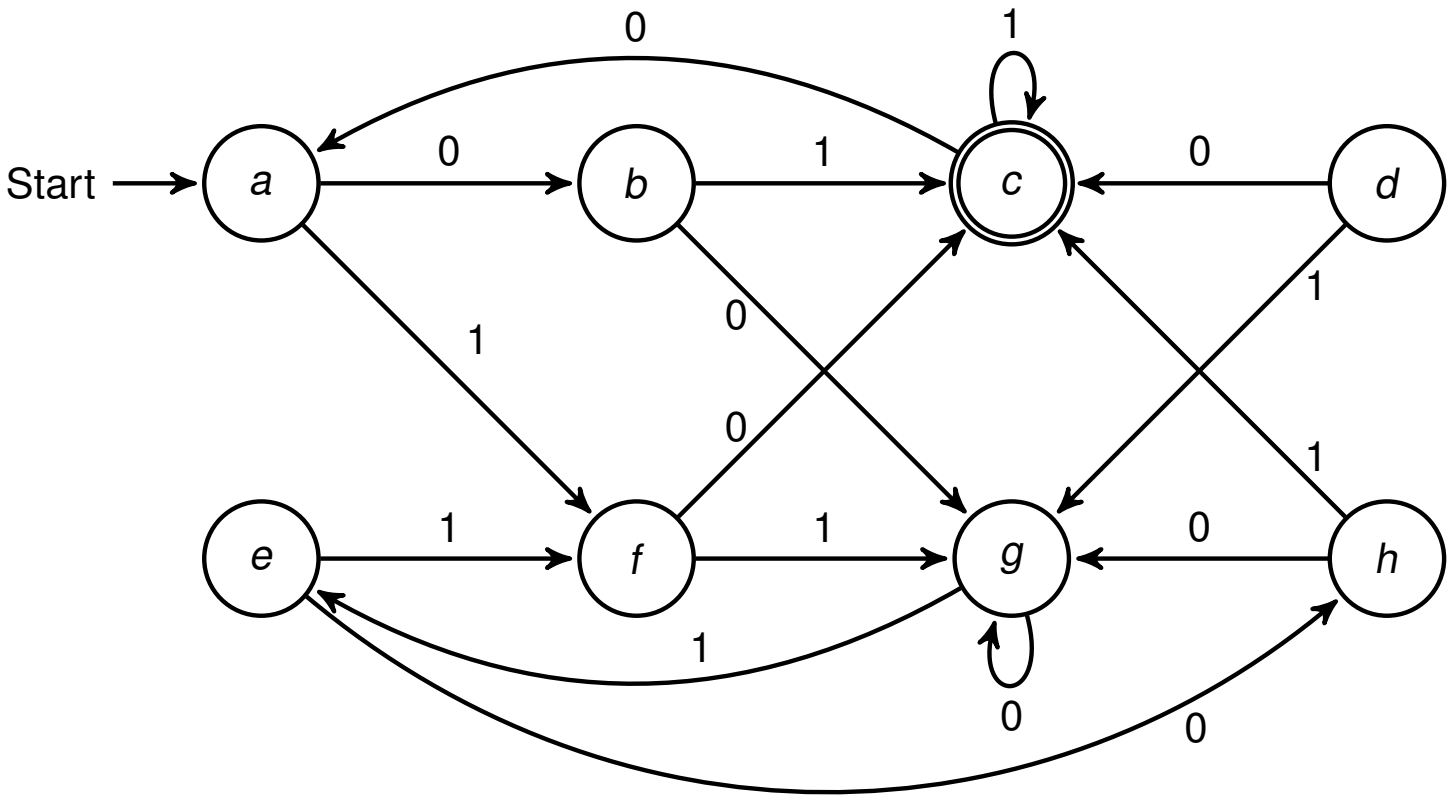
$$p \equiv q \Leftrightarrow \forall\, w \in \Sigma^* : \hat{\delta}(p, w) \in F \text{ iff } \hat{\delta}(q, w) \in F$$

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

$$\exists\, w \in \Sigma^* : [\hat{\delta}(p, w) \in F \wedge \hat{\delta}(q, w) \notin F] \vee [\hat{\delta}(p, w) \notin F \wedge \hat{\delta}(q, w) \in F].$$

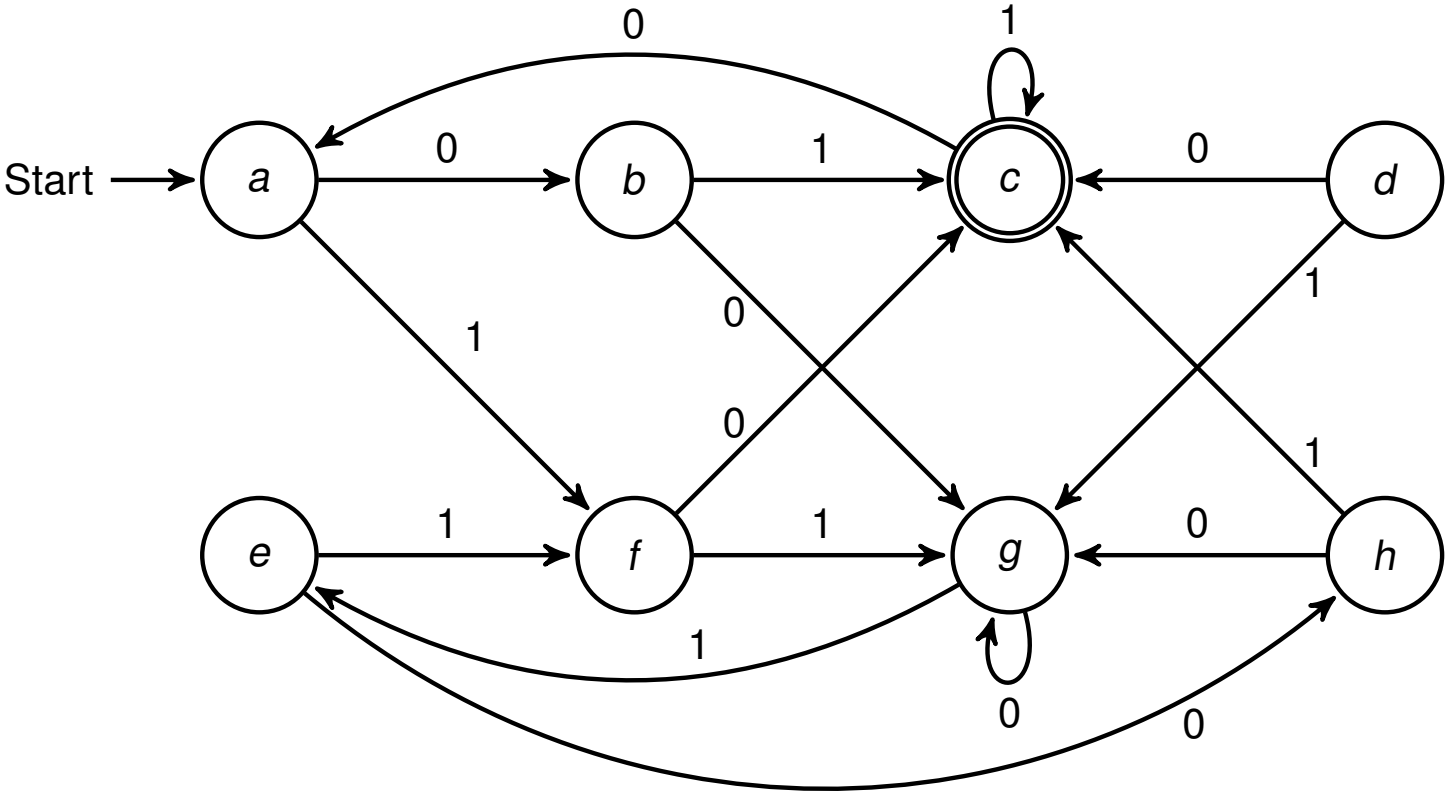
Example

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



Note that $\hat{\delta}(c, \epsilon) \in F$ and $\hat{\delta}(g, \epsilon) \notin F$ imply $c \not\equiv g$.

Also, $\hat{\delta}(a, 01) = c \in F$ and $\hat{\delta}(g, 01) = e \notin F$ imply $a \not\equiv g$.

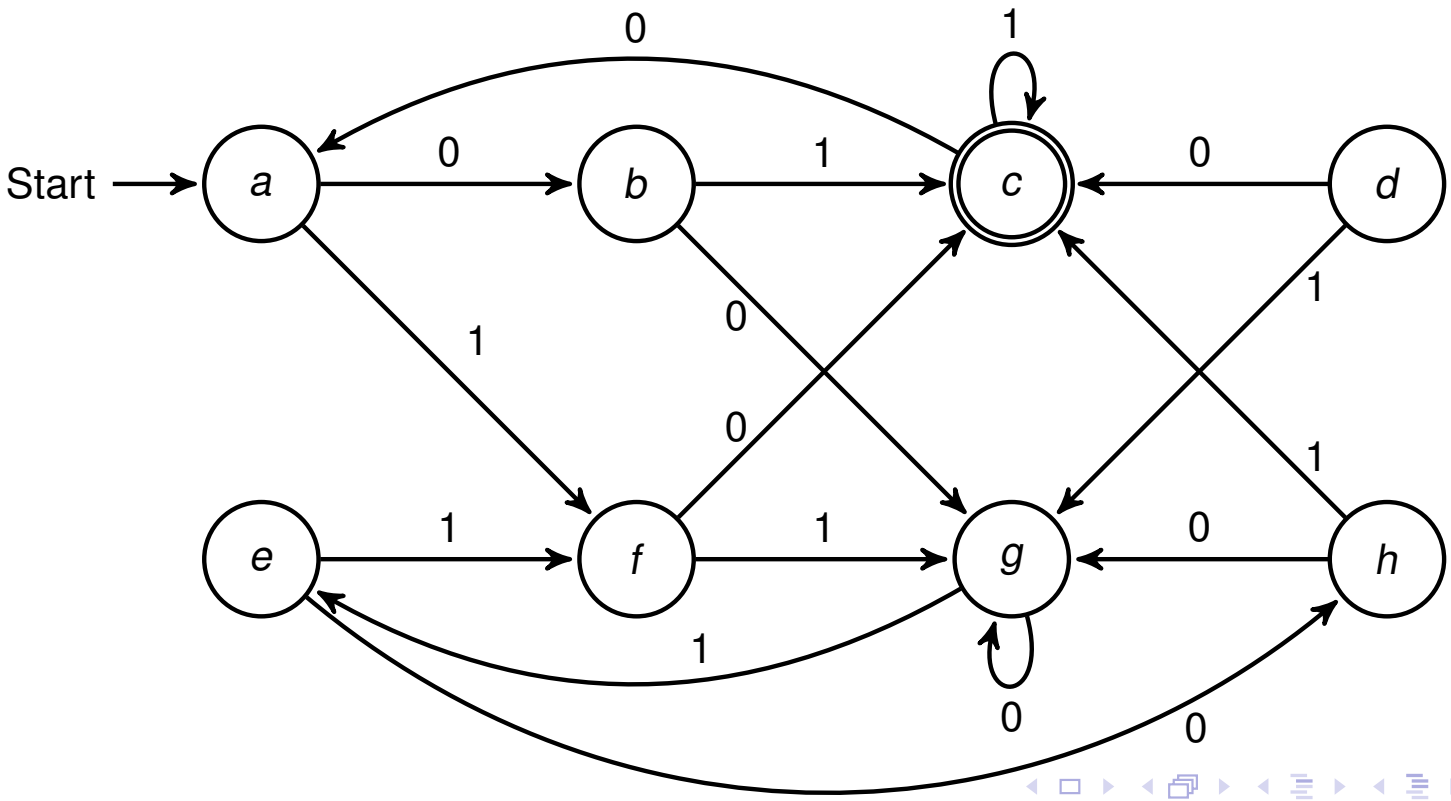


What about a and e?

Since

- $\hat{\delta}(a, \epsilon) = a \notin F$ and $\hat{\delta}(e, \epsilon) = e \notin F$,
- $\hat{\delta}(a, 1) = f = \hat{\delta}(e, 1)$ implies $\hat{\delta}(a, 1x) = \hat{\delta}(f, x) = \hat{\delta}(e, 1x)$,
- $\hat{\delta}(a, 0) = b \notin F$ and $\hat{\delta}(e, 0) = h \notin F$,
- $\hat{\delta}(a, 00) = g = \hat{\delta}(e, 00)$, and
- $\hat{\delta}(a, 01) = c = \hat{\delta}(e, 01)$;

we can conclude: $a \equiv e$.



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

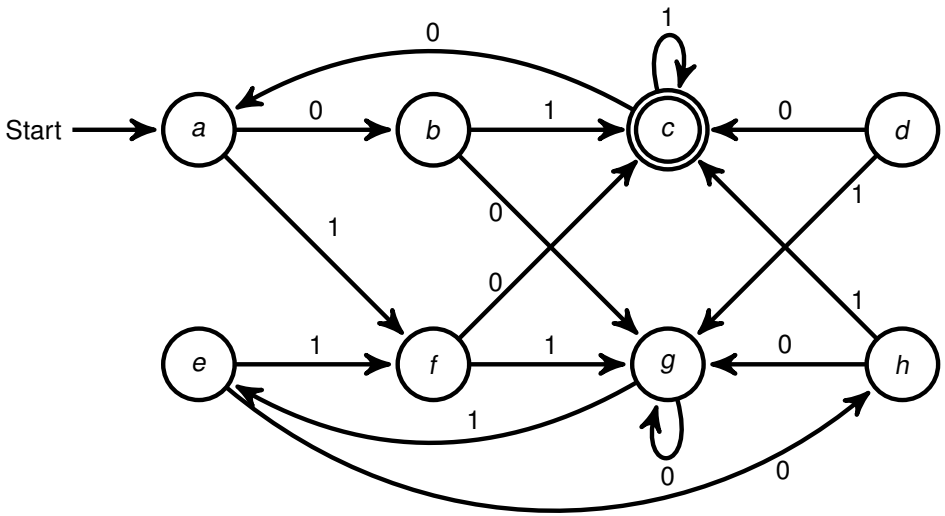
Basis step: If $p \in F$ and $q \notin F$, then $p \not\equiv q$.

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

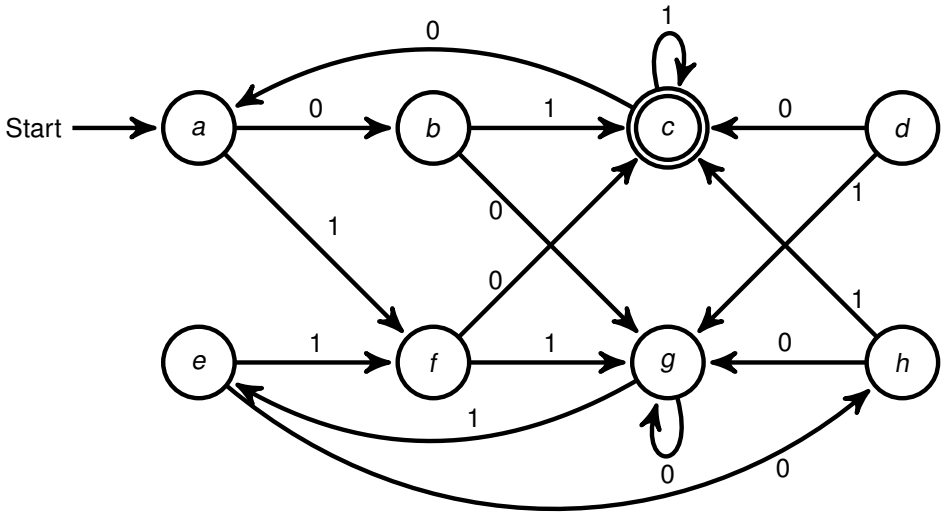
The reason of the induction rule is that there must be some string w such that exactly one of $\hat{\delta}(r, w)$ and $\hat{\delta}(s, w)$ is accepting. Then string aw must distinguish p from q.

Example

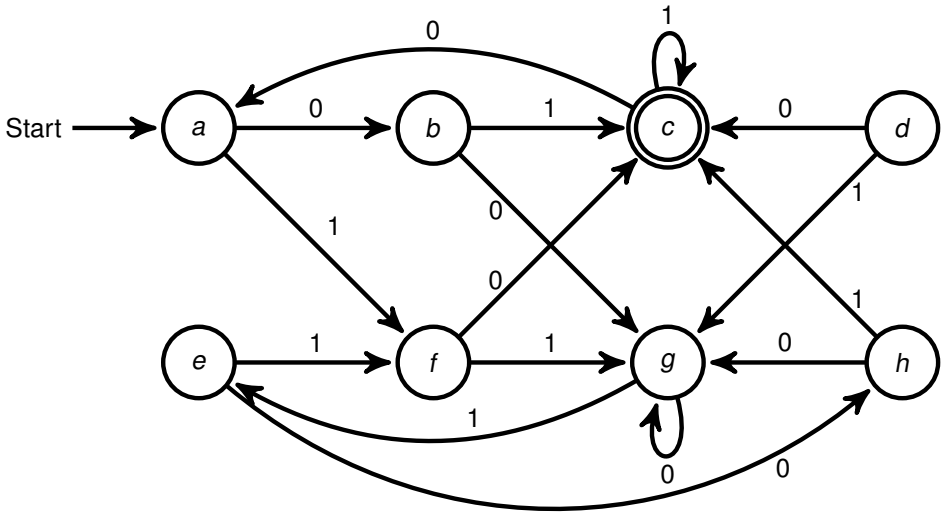
Apply the TF-algorithm to A



	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 the 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 $\hat{\delta}(q, w) \notin F$;
- while 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 $w = a_1 a_2 \cdots a_n$ be the shortest string that identifies some bad pair, and $\{p, q\}$ a bad pair that is identified by w and cannot be identified by any string shorter than w.

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 otherwise $\{r, s\}$ would be identified by any string shorter than w. However, the TF-algorithm have discovered that r and s are distinguishable by $a_2 a_3 \cdots a_n$.

Hence the TF-algorithm would distinguish p from q in the inductive step, and thus the theorem follows. □

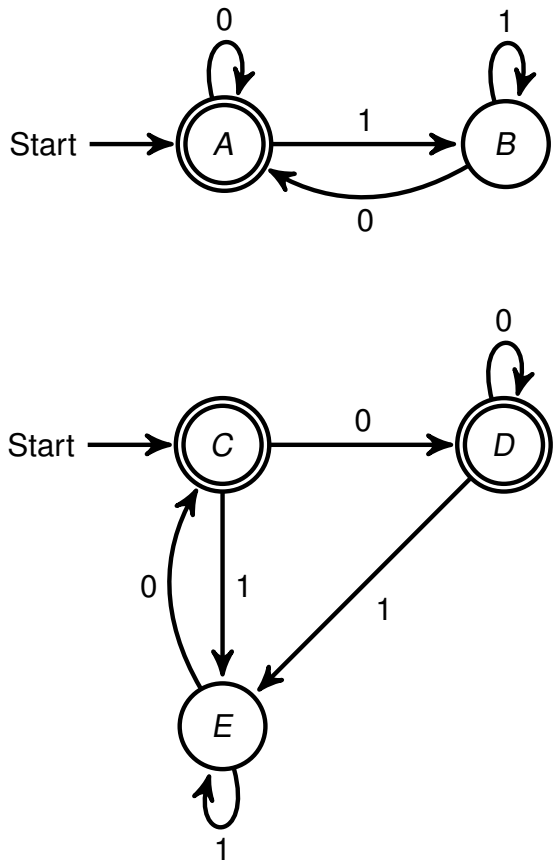
Testing Equivalence of Regular Languages

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 the TF-algorithm says that the two start states are distinguishable, then $L \neq M$, otherwise $L = M$.

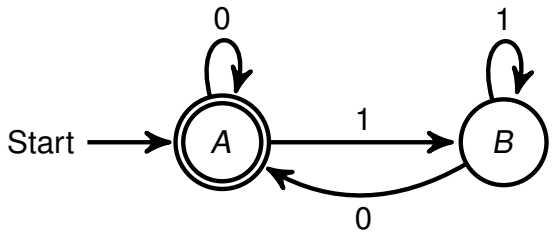
Example

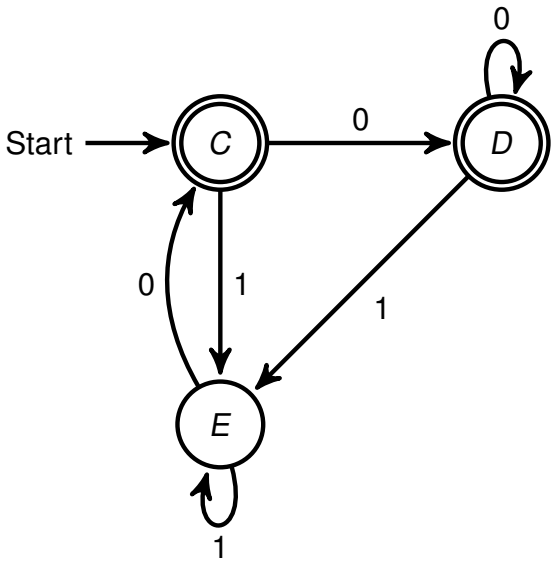
Consider the two DFA.

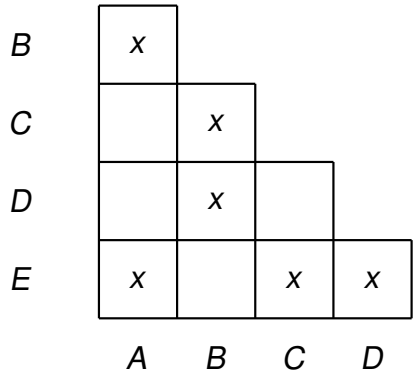


We can see that both DFA accept the language $L(\epsilon + (\mathbf{0} + \mathbf{1})^*\mathbf{0})$.

The result of the TF-algorithm is







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

Minimization of Automata

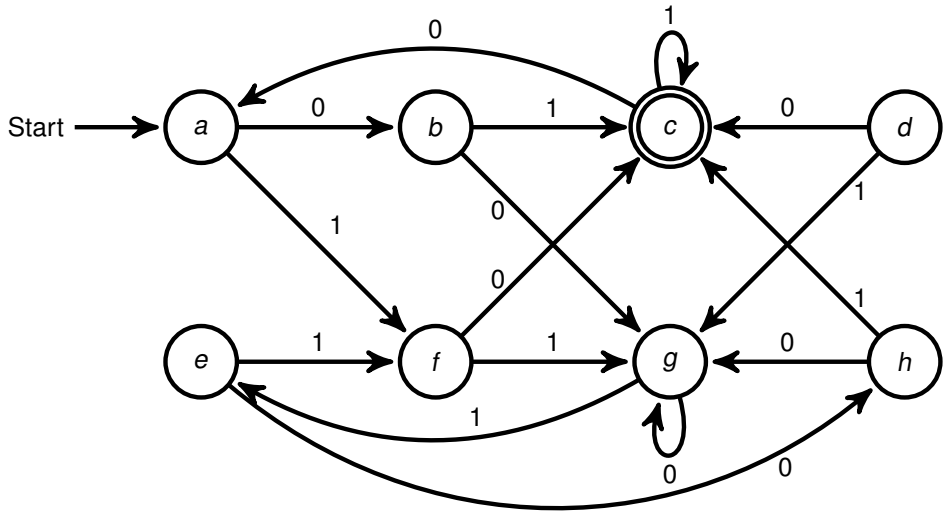
Minimization of DFA

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 cannot 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.

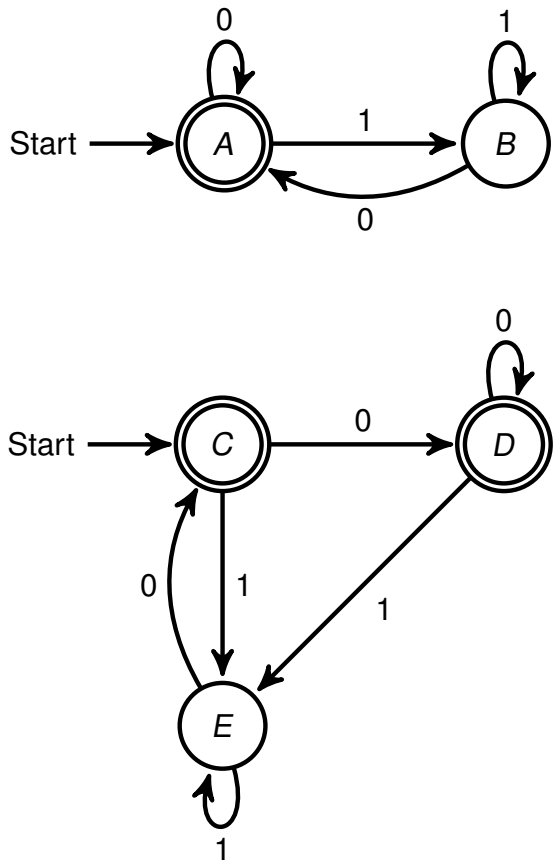
Example

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.

Corollary

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

However, $\hat{\delta}(q, w)$ is either accepting state or not. If $\hat{\delta}(q, w) \in F$, then $q \not\equiv r$; and if $\hat{\delta}(q, w) \notin F$, then $q \not\equiv p$. The vice verse case is symmetrical.

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

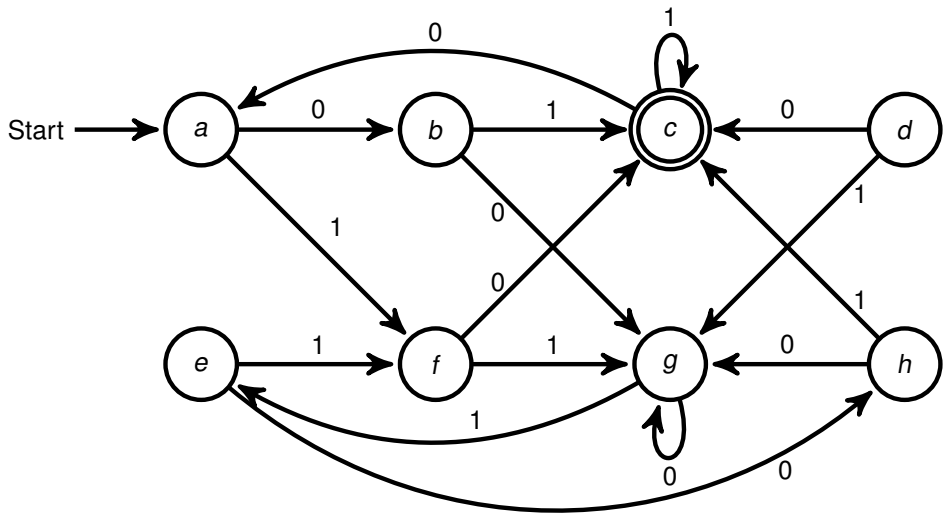
1. Use the TF-algorithm to find all the pairs of equivalent states.
2. Partition the set of states Q into blocks of equivalent states.
3. Construct the minimum-state equivalent DFA $B = ([Q]_{\equiv}, \Sigma, \gamma, [q_0]_{\equiv}, [F]_{\equiv})$, where

$$\gamma([q]_{\equiv}, a) = [\delta(q, a)]_{\equiv}$$

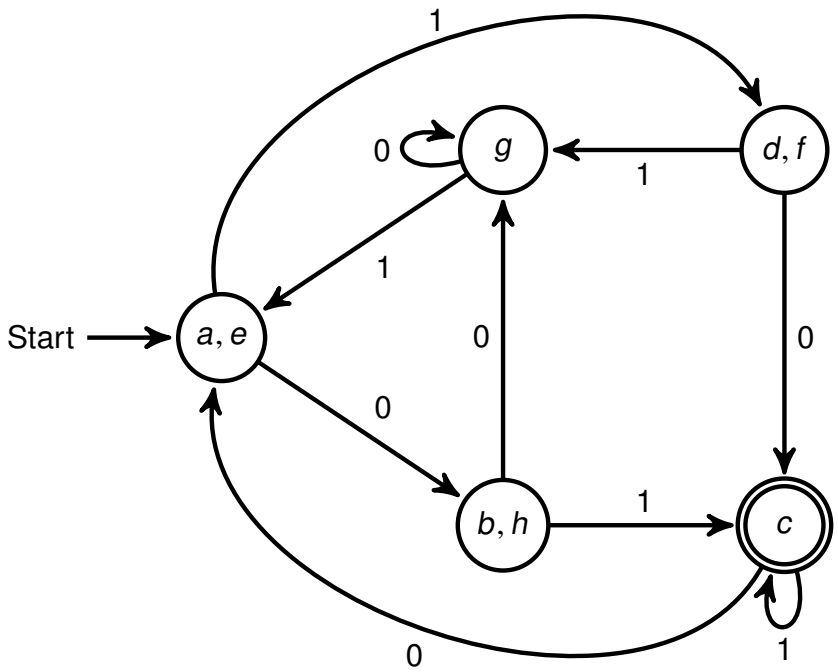
In order to define B well, we have to show "if $p \equiv q$ then $\delta(p, a) \equiv \delta(q, a)$". By the TF-algorithm, we know it is true. Note also that $[F]_{\equiv}$ may contain one or more block of equivalent accepting states of A.

Example

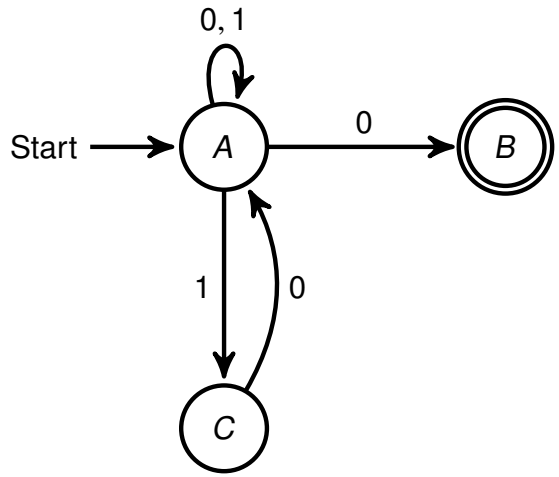
We can minimize DFA A.



The minimum-state DFA B is equivalent with A.



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



Homework

Exercises 4.4.1 & 4.4.3

Suppose that p and q are distinguishable states of a given DFA A with n states. As a function of n, what is the tightest upper bound on how long the shortest string that distinguishes p from q can be?