Formal Language and Automata Theory

Chapter Two

Finite automata (FA)

DFA Vs NFA

- When the machine is in a given state and reads the next input symbol, we know what the next state will be – it is determined.
- In nondeterministic machine, several choices may exist for the next state at any point.
- Non-determinism is a generalization of determinism, so every *deterministic finite automaton* is automatically a *non-deterministic finite automaton.*
- DFAs are clearly a subset of NFAs.

Deterministic Finite Automaton (DFA)

Formal Definition of a DFA

A DFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- **Q** is a finite set of states.
- **Σ** is a finite set of symbols called the input alphabet.
- **δ** is the transition function where $\delta: Q \times \Sigma \rightarrow Q$
- **q_0** is the initial state from where any input is processed $(q_0 \in Q)$.
- **F** is a set of final state/states of Q $(F \subseteq Q)$.

Con’t

Graphical Representation of a DFA

A DFA is represented by digraphs called **state diagram**:

- The vertices represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

Con’t

Example

Let a deterministic finite automaton be

- $Q = \{a, b, c\}$,
- $\Sigma = \{0, 1\}$,
- $q_0=\{a\}$,
- $F=\{c\}$, and

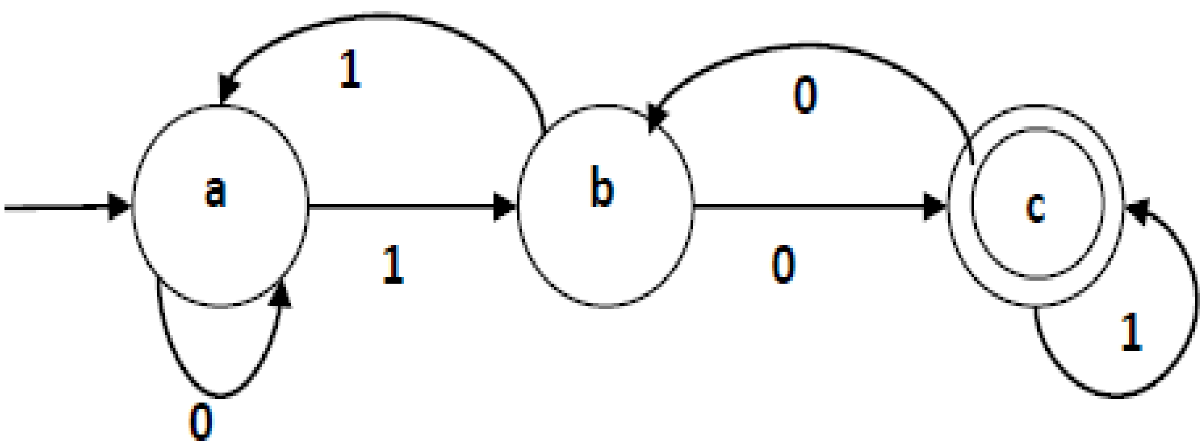
Con’t

- Transition function δ as shown by the following table:

Present state	Next state for input 0	Next state for input 1
a	a	b
b	c	a
c	b	c

Con’t

- Its graphical representation would be as follows:



Con’t

- Find the result of $\delta(a, 0101)$

Solution:

$\delta(a, o1o1)$

$= \delta(\delta(a,0), 101)$

$=\delta (a, 101)$

$=\delta (\delta(a,1), 01)$

$=\delta(b, 01)$

$=\delta(\delta(b,0),1)$

$=\delta(c,1)$

$=c$

Non-deterministic Finite Automaton (NDFA)

Formal Definition of an NDFA

An NDFA can be represented by a 5-tuple (Q, Σ, δ, q_0, F) where:

- **Q** is a finite set of internal states.
- **Σ** is a finite set of symbols called the input alphabets.
- **δ** is the transition function where δ: Q × {Σ U ε} → 2Q

(Here the power set of Q (2Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)

- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q (F ⊆ Q).

Con’t

Graphical Representation of an NDFA: (same as DFA)

An NDFA is represented by digraphs called state diagram.

- The vertices represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

Con’t

Example

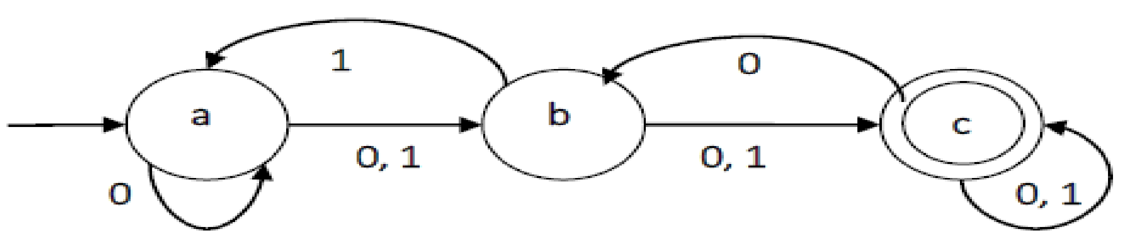
- Let a non-deterministic finite automaton be $\longrightarrow$
- $Q = \{a, b, c\}$
- $\Sigma = \{0, 1\}$
- $q_0 = \{a\}$
- $F=\{c\}$

Con’t

The transition function δ as shown below:

Present State	Next State for Input 0	Next State for Input 1
a	a, b	b
b	c	a, c
c	b, c	c

- Its graphical representation would be as follows:



Converting an NDFA to an Equivalent DFA

- Let $\mathbf{X = (Q_x, \Sigma, \delta_x, q_0, F_x)}$ be an NDFA which accepts the language L(X). We have to design an equivalent DFA $\mathbf{Y = (Q_y, \Sigma, \delta_y, q_0, F_y)}$ such that **L(Y) = L(X)**.

Con’t

The following procedure converts the NDFA to its equivalent DFA:

Algorithm:

Input: An NDFA

Output: An equivalent DFA

- **Step 1** Create state table from the given NDFA.
- **Step 2** Create a blank state table under possible input alphabets for the equivalent DFA.
- **Step 3** Mark the start state of the DFA by q_0 (Same as the NDFA).

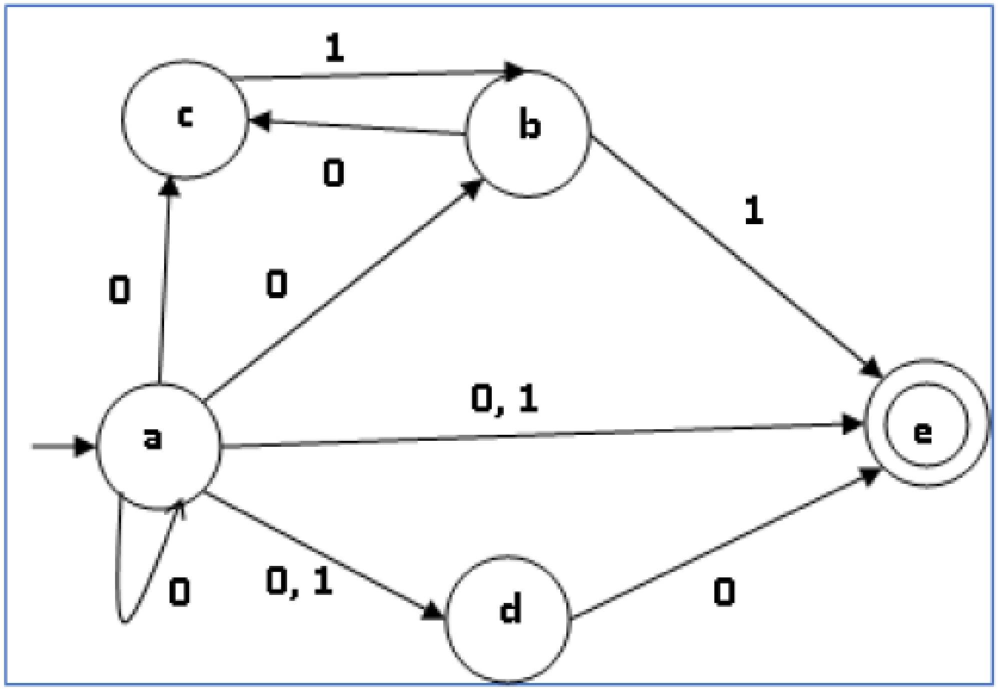
Con’t

- **Step 4** Find out the combination of States $\{Q_0, Q_1, \ldots, Q_n\}$ for each possible input alphabet.
- **Step 5** Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.
- **Step 6** The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

Con’t

Example 1

- Let us consider the NDFA shown in the figure below.



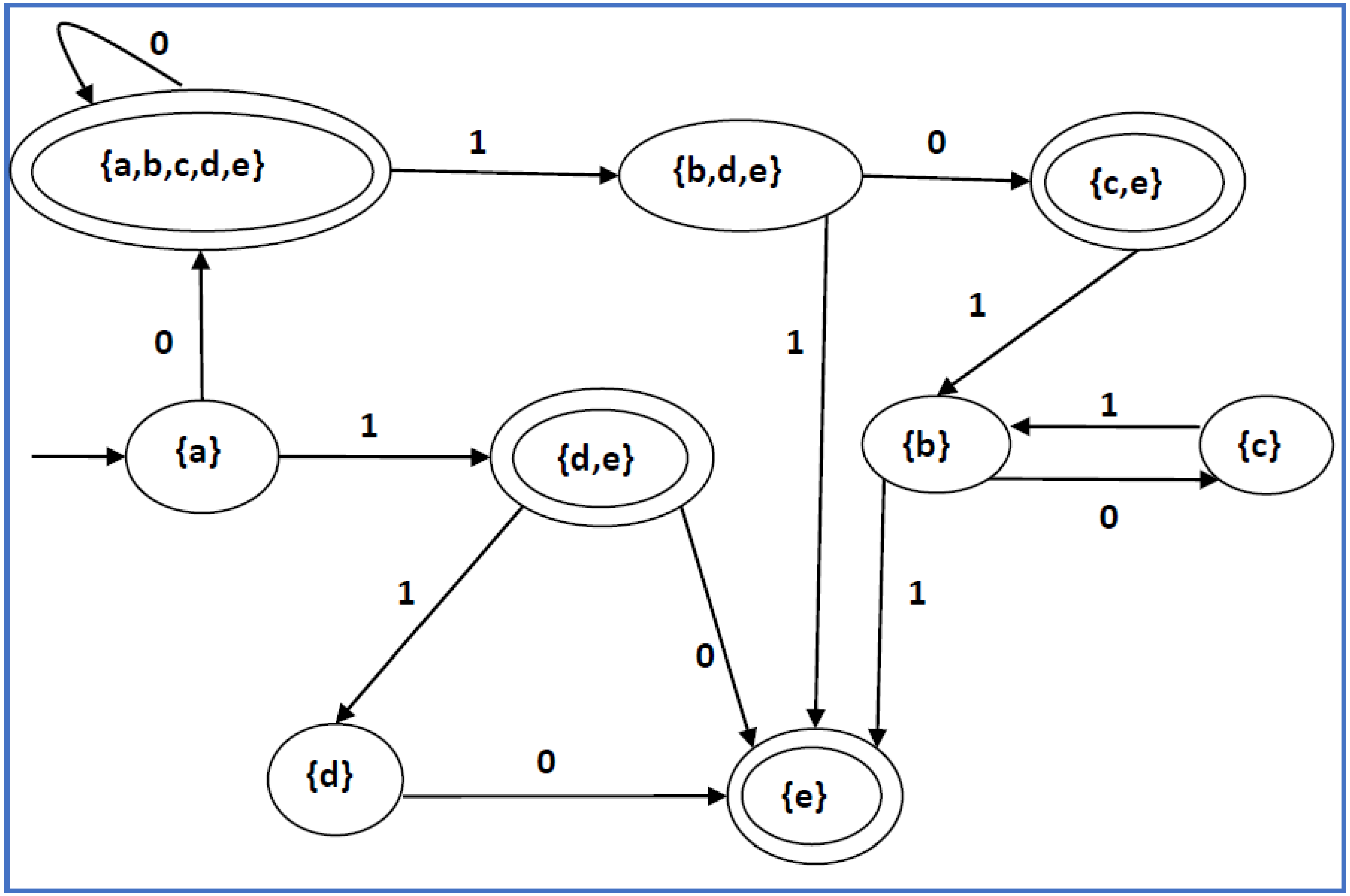
q	δ(q,0)	δ(q,1)
a	{a,b,c,d,e}	{d,e}
b	{c}	{e}
c	∅	{b}
d	{e}	∅
e	∅	∅

Con’t

q	δ(q,0)	δ(q,1)
a	{a,b,c,d,e}	{d,e}
{a,b,c,d,e}	{a,b,c,d,e}	{b,d,e}
{d,e}	e	D
{b,d,e}	{c,e}	E
e	∅	∅
d	e	∅
{c,e}	∅	B
b	c	E
c	∅	B

State table of DFA equivalent to NDFA

Con’t

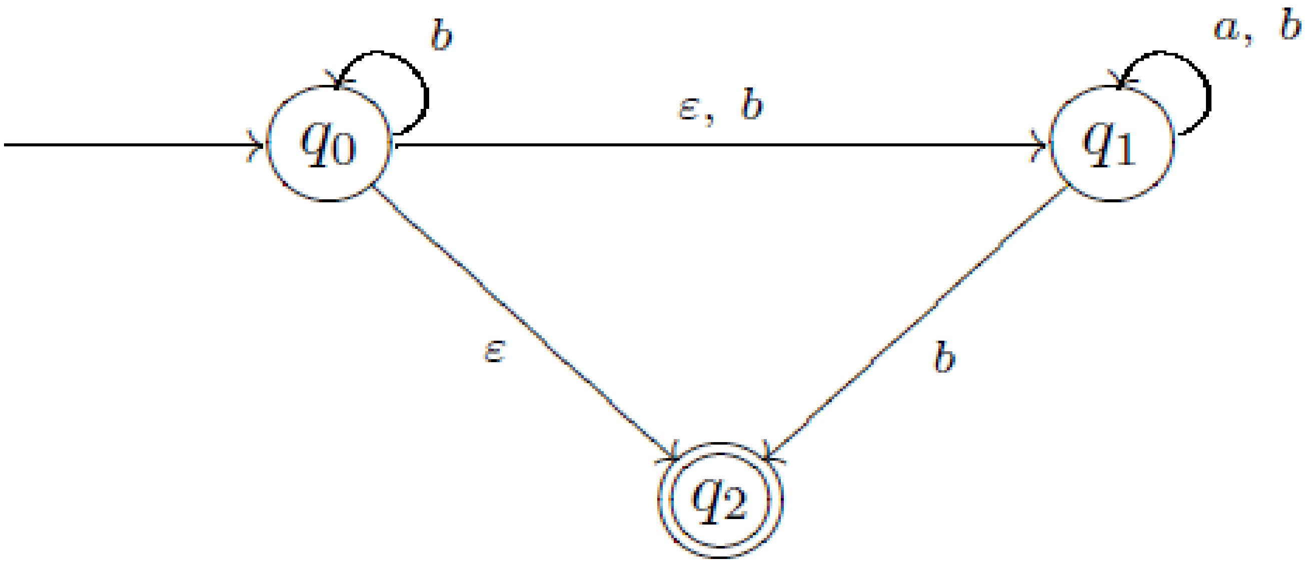


State diagram of DFA

Con't

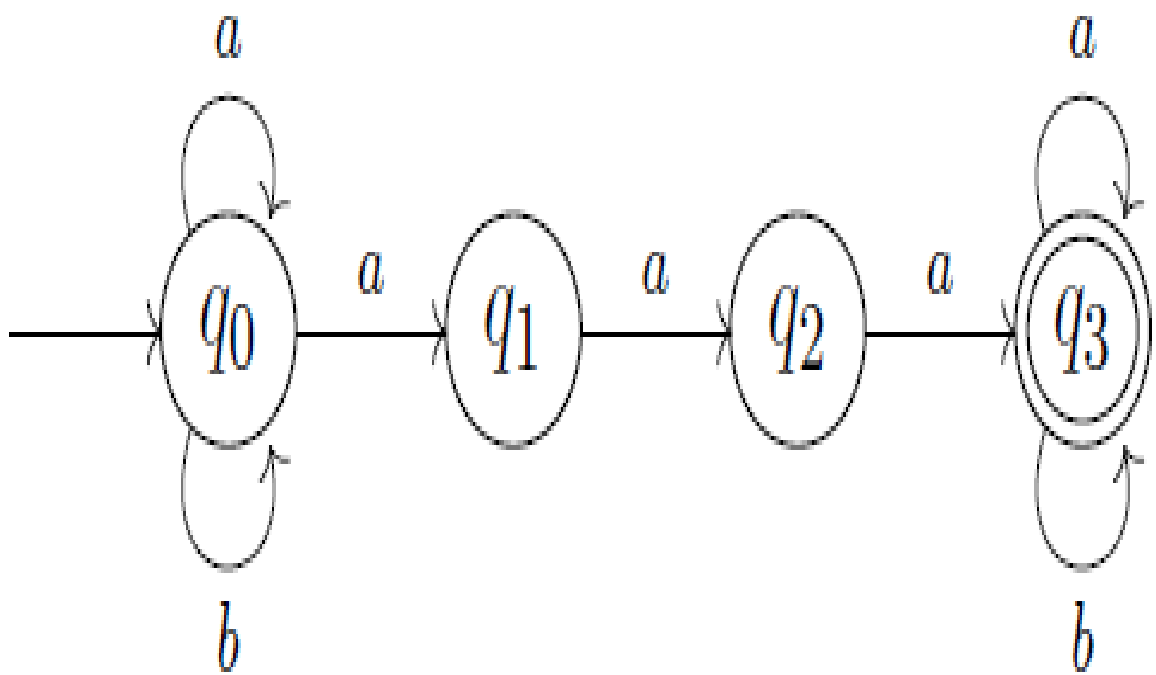
Exercise

Construct an equivalent DFA for a given NDFA



Assignment

Construct an equivalent DFA for a given NDFA

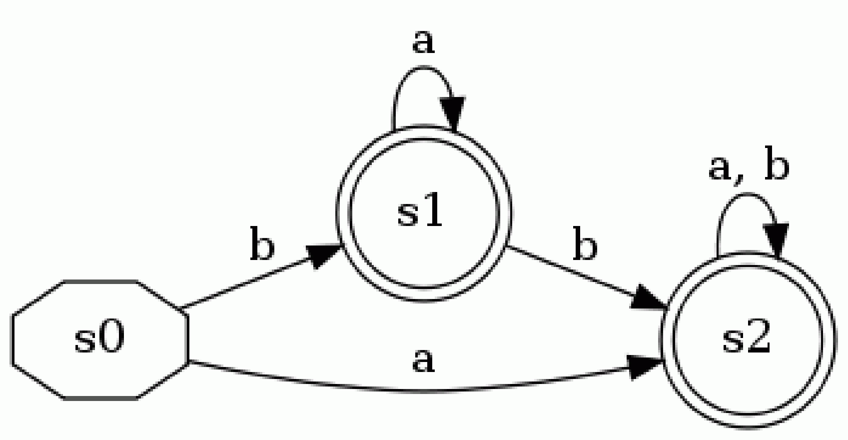


DFA Minimization

- Questions of DFA size:
  - Given a DFA, can we find one with fewer states that accepts the same language?
  - What is the smallest DFA for a given language?
  - Is the smallest DFA unique, or can there be more than one "smallest" DFA for the same language?
- All these questions have neat answers…
- The task of *DFA minimization*, then, is to automatically transform a given DFA into a state-minimized DFA
  - Several algorithms and variants are known
  - Note that this also in effect can minimize an NFA (since we know algorithm to convert NFA to DFA)

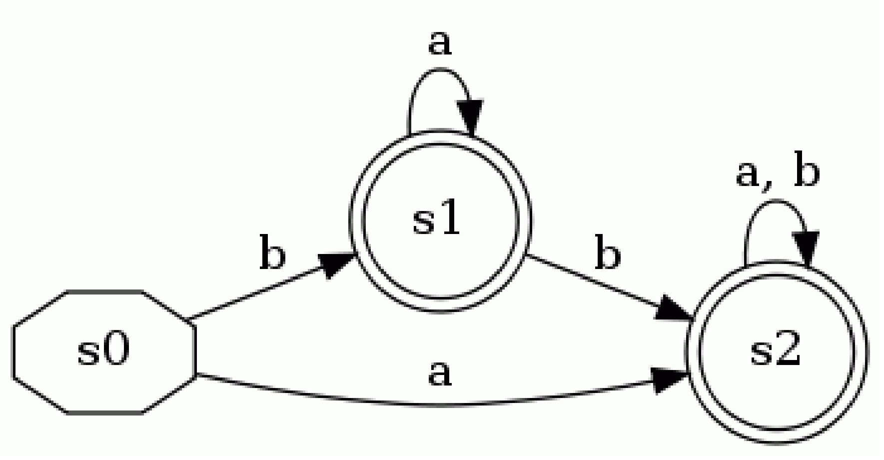
DFA Minimization

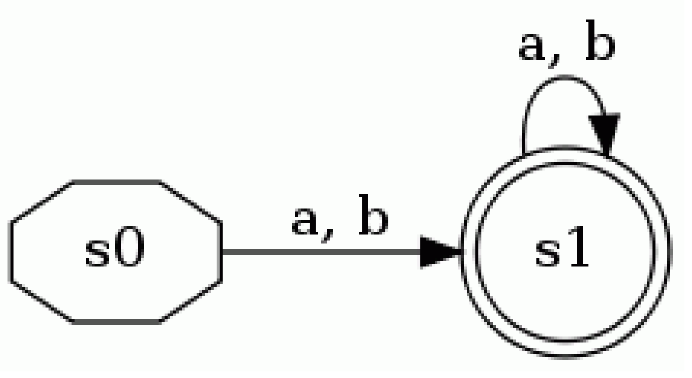
- Some states can be redundant:
  - The following DFA accepts **(a|b)+**
  - State s1 is not necessary



DFA Minimization

- So these two DFAs are *equivalent*:





State Reduction by Partitioning

- We say two states **p** and **q** are **equivalent** (or indistinguishable), if, for every string w $\in \Sigma^*$, transition $\delta($ ***p*** , *w) ends in an accepting state if and only if* $\delta($ ***q*** , *w) does*. In the preceding slide states S_1 and S_2 are equivalent.
- There are efficient algorithms available for computing the sets of equivalent states of a given DFA.
- The following two slides show:
  - the detailed steps for computing equivalent state sets of the DFA
  - constructing the reduced DFA.

State Reduction by Partitioning

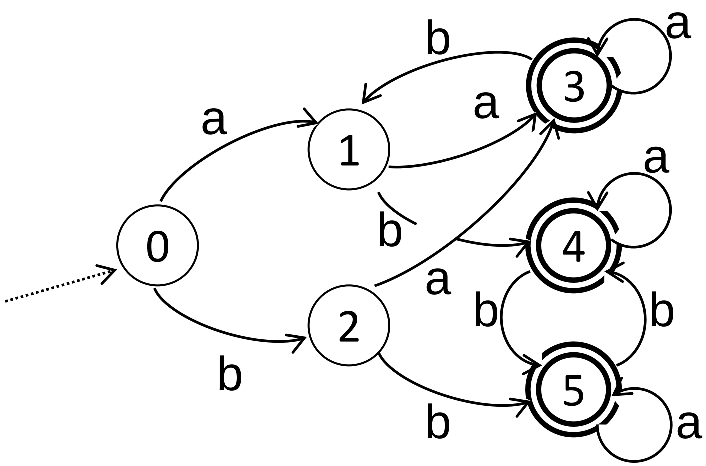


Figure (a) A DFA

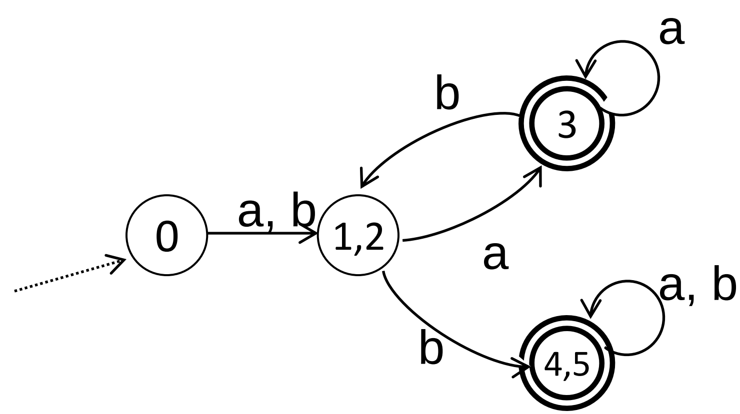


Figure (b) Reduced DFA

- **Step 0**: Partition the states according to accepting/non-accepting.

P_1	P_2
{ 3, 4, 5 }	{ 0, 1, 2 }

State Reduction by Partitioning(cont'ed)

- **Step 1**: Get the response of each state for each input symbol. Notice that States 3 and 0 show different responses from the ones of the other states in the same set.

$$
\begin{array}{lccc}
\multicolumn{4}{c}{\boldsymbol{P_1}} \\
\hline
 & p_1 & p_1 & p_1 \\
\mathbf{a}\rightarrow & \uparrow & \uparrow & \uparrow \\
 & \{3, & 4, & 5\} \\
\mathbf{b}\rightarrow & \downarrow & \downarrow & \downarrow \\
 & p_2 & p_1 & p_1
\end{array}
\qquad\qquad
\begin{array}{lccc}
\multicolumn{4}{c}{\boldsymbol{P_2}} \\
\hline
 & p_2 & p_1 & p_1 \\
\mathbf{a}\rightarrow & \uparrow & \uparrow & \uparrow \\
 & \{0, & 1, & 2\} \\
\mathbf{b}\rightarrow & \downarrow & \downarrow & \downarrow \\
 & p_2 & p_1 & p_1
\end{array}
$$

Record responses for each input symbol

• **Step 2**: Partition the sets according to the responses, and go to Step 1 until no partition occurs.

	P_{11}	P_{12}		P_{21}	P_{22}
	p_{11} p_{11}			p_{12} p_{12}	
a→	↑ ↑		a→	↑ ↑	
	{4, 5}	**{3}**		**{1, 2}**	**{0}**
b→	↓ ↓		b→	↓ ↓	
	p_{11} p_{11}			p_{11} p_{11}	

Partition the set, and record responses for each input symbol

• No further partition is possible for the sets $\boldsymbol{P_{11}}$ and $\boldsymbol{P_{21}}$. So the final partition results are as follows.

{4, 5} {3} {1, 2} {0}

Exercise

- Minimize the given DFA.