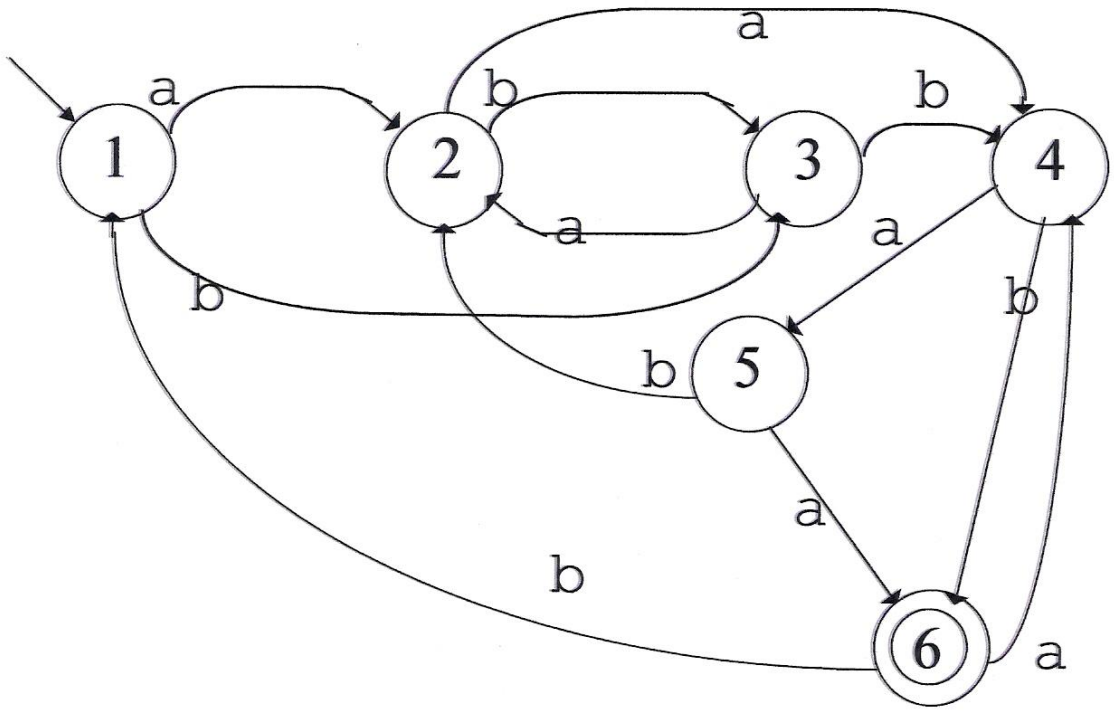
Finite State Machines
State Minimization

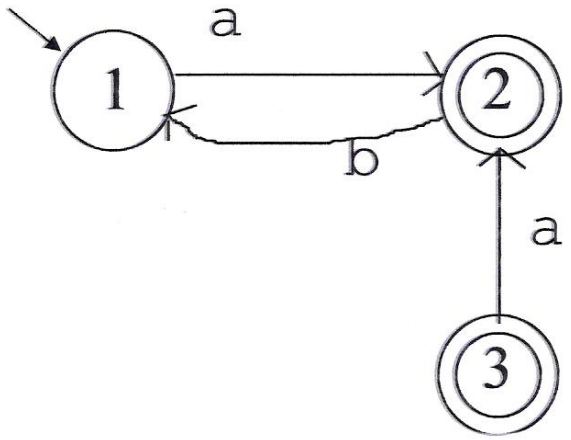
Consider:



Is this a minimal machine?

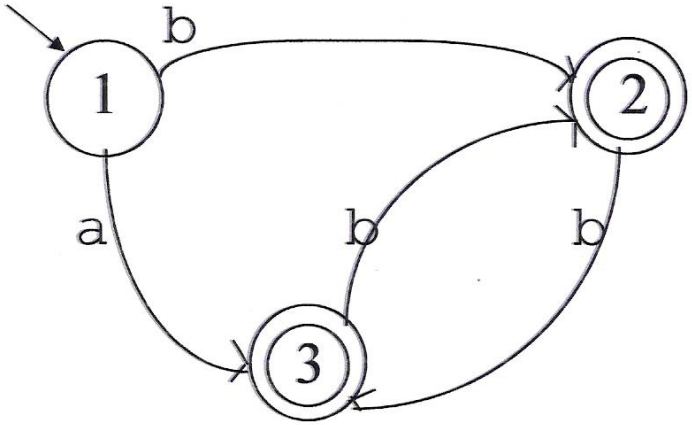
State Minimization

Step (1): Get rid of unreachable states.



State 3 is unreachable.

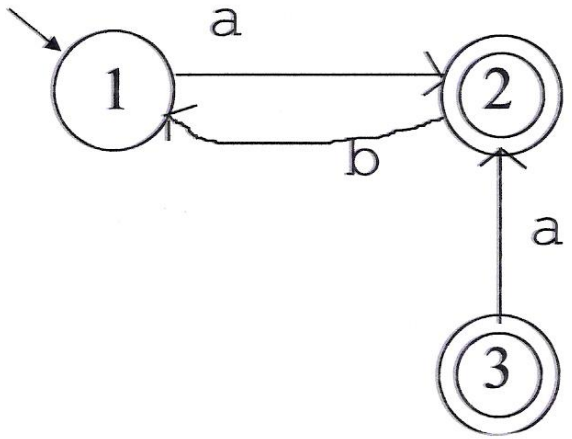
Step (2): Get rid of redundant states.



States 2 and 3 are redundant.

Removal of Unreachable States

We can't easily find the unreachable states directly. But we can find the reachable ones and determine the unreachable ones from there.



Finding an Algorithm for State Minimization

Capture the notion of equivalence classes of strings with respect to a language.

Prove that we can always find a (unique up to state naming) deterministic FSM with a number of states equal to the number of equivalence classes of strings.

Describe an algorithm for finding that deterministic FSM.

Defining Equivalence for Strings

We want to capture the notion that two strings are equivalent or ***indistinguishable*** with respect to a language *L* if, no matter what is tacked on to them on the right, either they will both be in *L* or neither will. Why is this the right notion? Because it corresponds naturally to what the states of a recognizing FSM have to remember.

Example:

(1) a b a b a b

(2) b a a b a b

Suppose $L = \{w \in \{a, b\}^* : |w| \text{ is even}\}$. Are (1) and (2) equivalent?

Suppose $L = \{w \in \{a, b\}^* : \text{every } a \text{ is immediately followed by } b\}$. Are (1) and (2) equivalent?

Defining Equivalence for Strings

If two strings are indistinguishable with respect to L, we write:

$$x \approx_L y$$

Formally, $x \approx_L y$ iff $\forall z \in \Sigma^* (xz \in L \text{ iff } yz \in L)$.

Defining Equivalence for Strings

$\approx_L$ is an equivalence relation because it is:

Reflexive: $\forall x \in \Sigma^* (x \approx_L x)$, because:

$$\forall x, z \in \Sigma^* (xz \in L \leftrightarrow xz \in L).$$

Symmetric: $\forall x, y \in \Sigma^* (x \approx_L y \rightarrow y \approx_L x)$, because:

$$\forall x, y, z \in \Sigma^* ((xz \in L \leftrightarrow yz \in L) \leftrightarrow (yz \in L \leftrightarrow xz \in L)).$$

Transitive: $\forall x, y, z \in \Sigma^* (((x \approx_L y) \wedge (y \approx_L w)) \rightarrow (x \approx_L w))$, because:

$$\forall x, y, z \in \Sigma^*$$
$$(((xz \in L \leftrightarrow yz \in L) \wedge (yz \in L \leftrightarrow wz \in L)) \rightarrow (xz \in L \leftrightarrow wz \in L)).$$

Defining Equivalence for Strings

because $i \approx_L$ is an equivalence relation :

No equivalence class of $\approx_L$ is empty.

Each string in Σ^* is in exactly one equivalence class of $\approx_L$.

Defining Equivalence for Strings

Example:

$\Sigma = \{a, b\}$

$L = \{w \in \Sigma^*$: every a is immediately followed by $b\}$

The equivalence classes of $\approx_L$:

[1] [ε, b, abb, …] [all strings in L].

[2] [a, $abba$, …] [all strings that end in a and have no prior a that is not followed by a b].

[3] [aa, $abaa$, …] [all strings that contain at least one instance of aa].

Defining Equivalence for Strings

Example:

$\Sigma = \{a, b\}$

$L = \{w \in \Sigma^* : |w| \text{ is even}\}$

The equivalence classes of $\approx_L$:

[1] [ε, aa, ab, …] [all strings in *L*].

[2] [a, b, …] [all strings not in L].

Defining Equivalence for Strings

Example:

$\Sigma = \{a, b\}$
$L = \{w \in \Sigma^* :$ no two adjacent characters are the same$\}$

The equivalence classes of $\approx_L$:

[1] [ε]
[2] [a, aba, ababa, …]
[3] [b, ab, bab, abab, …]
[4] [aa, abaa, ababb…]

Defining Equivalence for Strings

Example:

$\Sigma = \{a, b\}$
$L = \{a^n b^n, n \geq 0\}$

The equivalence classes of $\approx_L$:
Consider strings:

ε	aa	aaaa
a	aba	aaaaa
b	aaa	

...

DFSM and Equivalence Classes of Strings

***Theorem*:** Let L be a regular language and let M be a DFSM that accepts L. The number of states in M is greater than or equal to the number of equivalence classes of $\approx_L$.

***Proof*:** Suppose that the number of states in M were less than the number of equivalence classes of $\approx_L$. Then, by the pigeonhole principle, there must be at least one state q that contains strings from at least two equivalence classes of $\approx_L$. But then M's future behavior on those strings will be identical, which is not consistent with the fact that they are in different

DFSM and Equivalence Classes of Strings

Theorem: Let L be a regular language over some alphabet Σ. Then there is a DFSM M that accepts L and that has precisely n states where n is the number of equivalence classes of $\approx_L$. Any other FSM that accepts L must either have more states than M or it must be equivalent to M except for state names.

DFSM and Equivalence Classes of Strings

Proof: (by construction)

$M = (K, \Sigma, \delta, s, A)$, where:

- K contains n states, one for each equivalence class of $\approx_L$.
- $s = [\varepsilon]$, the equivalence class of ε under $\approx_L$.
- $A = \{[x] : x \in L\}$.
- $\delta([x], a) = [xa]$.

(In other words, if M is in the state that contains some string x, then, after reading the next symbol, a, it will be in the state that contains xa.)

DFSM and Equivalence Classes of Strings

Proof, Continued

We must show that:

- K is finite. Since L is regular, it is accepted by some DFSM M'. M' has some finite number of states m. By Theorem 5.4, $n \leq m$. So K is finite.

- δ is a function. In other words, it is defined for all (state, input) pairs and it produces, for each of them, a unique value. The construction defines a value of δ for all (state, input) pairs. The fact that the construction guarantees a unique such value follows from the definition of $\approx_L$.

DFSM and Equivalence Classes of Strings

Proof, Continued

$L = L(M)$. To prove this, we must first show that $\forall s, t\ (([\varepsilon], st) \vdash_M^* ([s], t))$. We do this by induction on $|s|$.

If $|s| = 0$ then we have $([\varepsilon], \varepsilon) \vdash_M^* ([\varepsilon], t)$, which is true since M simply makes zero moves.

DFSM and Equivalence Classes of Strings

Proof, Continued

Assume that the claim is true if $|s| = k$. Then we consider what happens when $|s| = k+1$. $|s| \geq 1$, so we can let $s = yc$ where $y \in \Sigma^*$ and $c \in \Sigma$. We have:

/* M reads the first k characters:
$([\varepsilon], yct) \vdash_M^* ([y], ct)$ (induction hypothesis, since $|y| = k$).

/* M reads one more character:
$([y], ct) \vdash_M^* ([yc], t)$ (definition of δ_M).

/* Combining those two, after M has read $k+1$ characters:
$([\varepsilon], yct) \vdash_M^* ([yc], t)$ (transitivity of $\vdash_M^*$).
$([\varepsilon], st) \vdash_M^* ([s], t)$ (definition of s as yc).

DFSM and Equivalence Classes of Strings

Proof, Continued

So we have :

[*] $\forall s, t\ (([\varepsilon], st) \vdash_M^* ([s], t))$.

Let t be ε. Let s be any string in Σ^*. By [*]:

$([\varepsilon], s) \vdash_M^* ([s], \varepsilon)$.

So M will accept s iff $[s] \in A$, which, by the way in which A was constructed, it will be if the strings in $[s]$ are in L. So M accepts precisely those strings that are in M.

DFSM and Equivalence Classes of Strings

Proof, Continued

There exists no smaller machine *M#* that also accepts *L*. This follows directly from Theorem 5.4, which says that the number of equivalence classes of $\approx_L$ imposes a lower bound on the number of states in any DFSM that accepts *L*.

There is no different machine *M#* that also has *n* states and that accepts *L*.

Constructing the Minimal DFA from $\approx_L$

$\Sigma = \{a, b\}$

$L = \{w \in \Sigma^* :$ no two adjacent characters are the same$\}$

The equivalence classes of $\approx_L$:

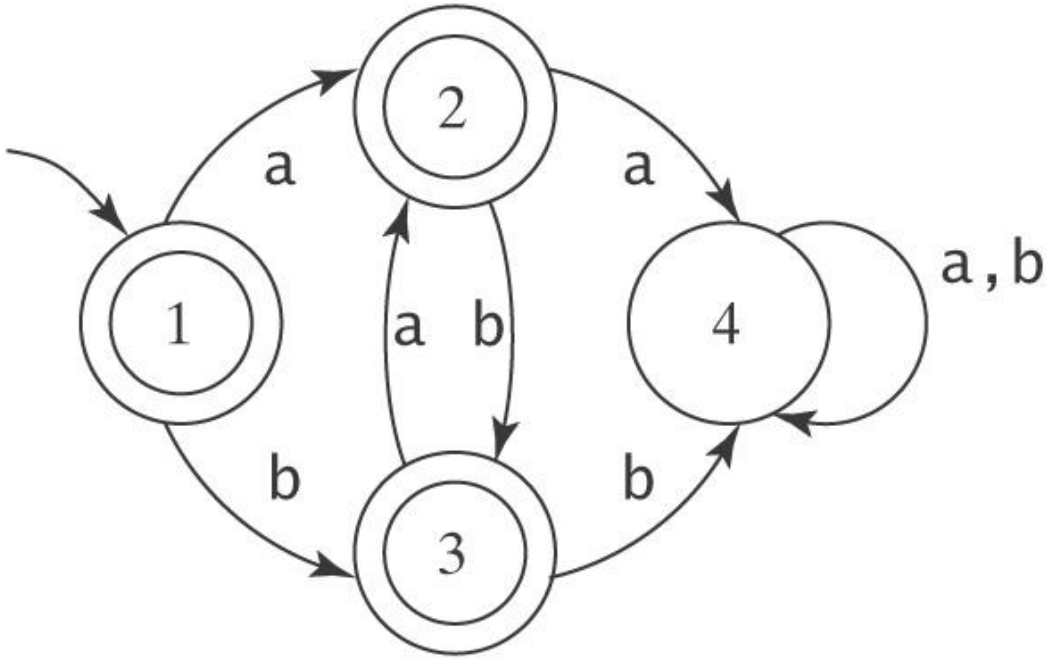
1: [ε]	ε
2: [a, ba, aba, baba, ababa, ...]	(b∪ε)(ab)*a
3: [b, ab, bab, abab, ...]	(a∪ε)(ba)*b
4: [bb, aa, bba, bbb, ...]	the rest

- Equivalence classes become states
- Start state is [ε]
- Accepting states are all equivalence classes in L
- $\delta([x], a) = [xa]$

Constructing the Minimal DFA from $\approx_L$

$\Sigma = \{a, b\}$

$L = \{w \in \Sigma^* : \text{no two adjacent characters are the same}\}$



The Myhill-Nerode Theorem

Theorem: A language is regular iff the number of equivalence classes of $\approx_L$ is finite.

Proof: Show the two directions of the implication:

L regular $\rightarrow$ the number of equivalence classes of $\approx_L$ is finite: If L is regular, then there exists some FSM M that accepts L. M has some finite number of states m. The cardinality of $\approx_L \leq m$. So the cardinality of $\approx_L$ is finite.

The number of equivalence classes of $\approx_L$ is finite $\rightarrow$ L regular: If the cardinality of $\approx_L$ is finite, then the construction that was described in the proof of the previous theorem will build an FSM that accepts L. So L must be regular.

Minimizing an Existing DFSM (Without Knowing $\approx L$)

Two approaches:

- Begin with *M* and collapse redundant states, getting rid of one at a time until the resulting machine is minimal.

- Begin by overclustering the states of *L* into just two groups, accepting and nonaccepting. Then iteratively split those groups apart until all the distinctions that *L* requires have been made.

The Overclustering Approach

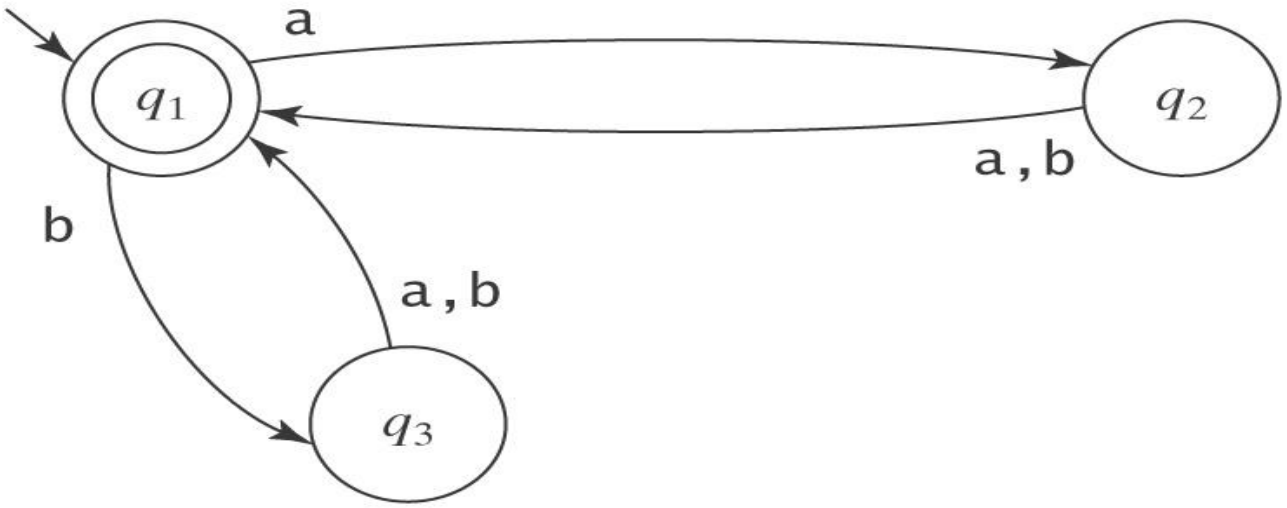
We need a definition for “equivalent”, i.e., mergeable states.

Define $q \equiv p$ iff for all strings $w \in \Sigma^*$, either w drives M to an accepting state from both q and p or it drives M to a rejecting state from both q and p.

The Overclustering Approach

An Example

$\Sigma = \{a, b\}$ $L = \{w \in \Sigma^* : |w| \text{ is even}\}$



$q_2 \equiv q_3$

The Overclustering Approach

Constructing $\equiv_n$

$p \equiv^0 q$ iff they behave equivalently when they read ε. In other words, if they are both accepting or both rejecting states.

$p \equiv^1 q$ iff they behave equivalently when they read any string of length 1, i.e., if any single character sends both of them to an accepting state or both of them to a rejecting state. Note that this is equivalent to saying that any single character sends them to states that are $\equiv^0$ to each other.

$p \equiv^2 q$ iff they behave equivalently when they read any string of length 2, which they will do if, when they read the first character they land in states that are $\equiv^1$ to each other. By the definition of $\equiv^1$, they will then yield the same outcome when they read the single remaining character.

And so forth.

The Overclustering Approach

Constructing ≡, Continued

More precisely, $\forall p,q \in K$ and any $n \geq 1$, $q \equiv^{n} p$ iff:

1. $q \equiv^{n-1} p$, and
2. $\forall a \in \Sigma\ (\delta(p, a) \equiv^{n-1} \delta(q, a))$

MinDFSM

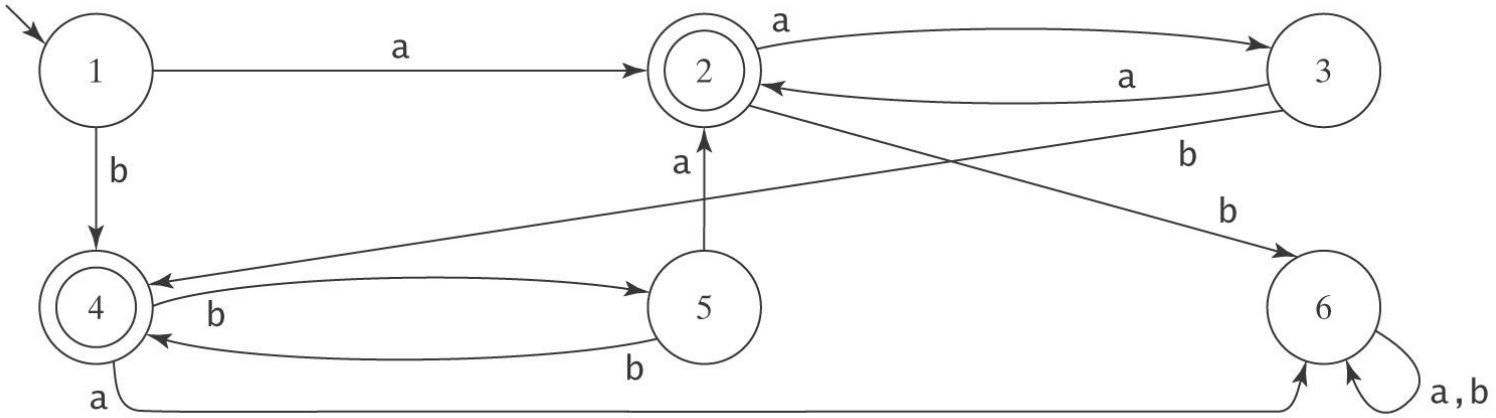
MinDFSM(*M*: DFSM) =

1. *classes* := {*A*, *K*-*A*};
2. Repeat until no changes are made
   - 2.1. *newclasses* := ∅;
   - 2.2. For each equivalence class *e* in *classes*, if *e* contains more than one state do
     - For each state *q* in *e* do
       - For each character *c* in Σ do
         - Determine which element of *classes* *q* goes to if *c* is read
     - If there are any two states *p* and *q* that need to be split, split them. Create as many new equivalence classes as are necessary. Insert those classes into *newclasses*.
     - If there are no states whose behavior differs, no splitting is necessary. Insert *e* into *newclasses*.
   - 2.3. *classes* := *newclasses*;
3. Return $M^* = (classes, \Sigma, \delta, [s_M], \{[q$: the elements of q are in $A_M]\})$, where δ_{M^*} is constructed as follows:
   - if $\delta_M(q, c) = p$, then $\delta_{M^*}([q], c) = [p]$

MinDFSM

An Example

$\Sigma = \{a, b\}$



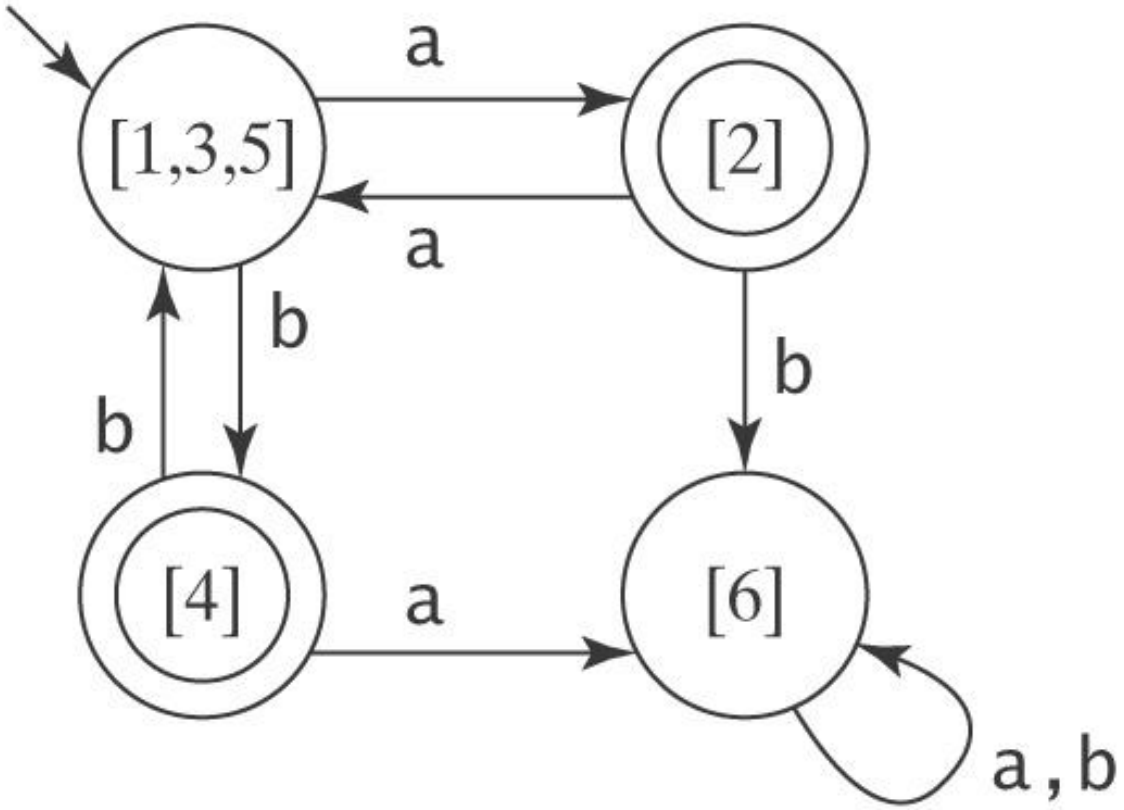
$\equiv^0 =$

$\equiv^1 =$

$\equiv^2 =$

MinDFSM

Result



Summary

- Given any regular language *L*, there exists a minimal DFSM *M* that accepts *L*.
- *M* is unique up to the naming of its states.
- Given any DFSM *M*, there exists an algorithm *minDFSM* that constructs a minimal DFSM that also accepts *L*(*M*).