



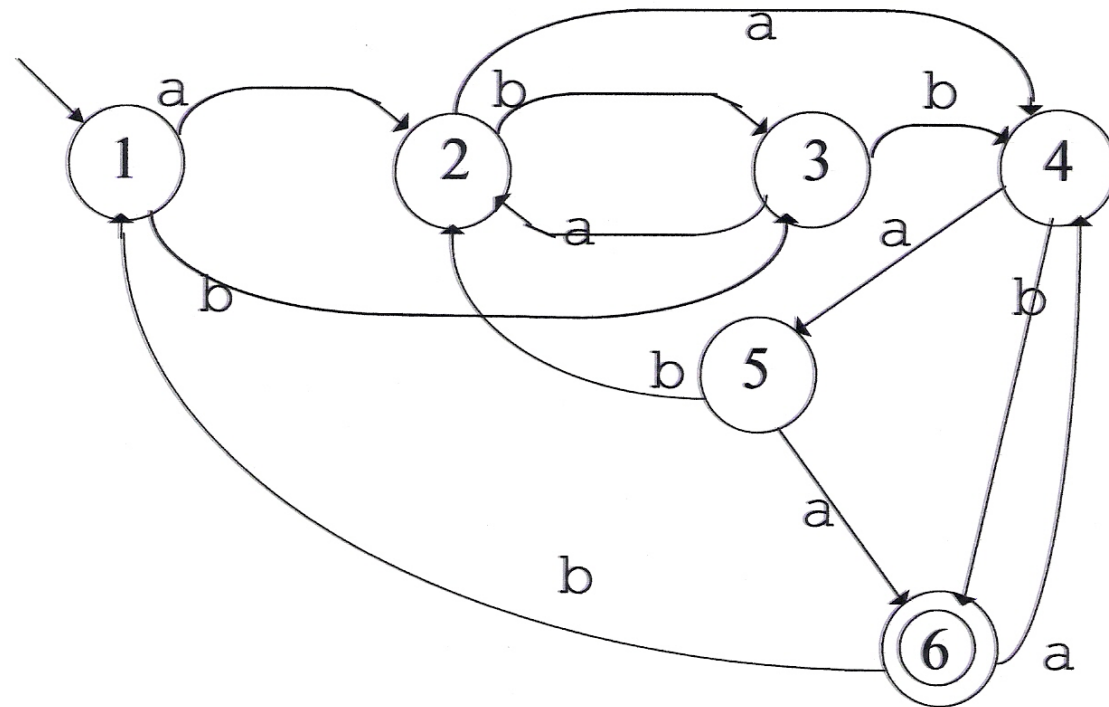
Finite State Machines

State Minimization

Chapter 5

State Minimization

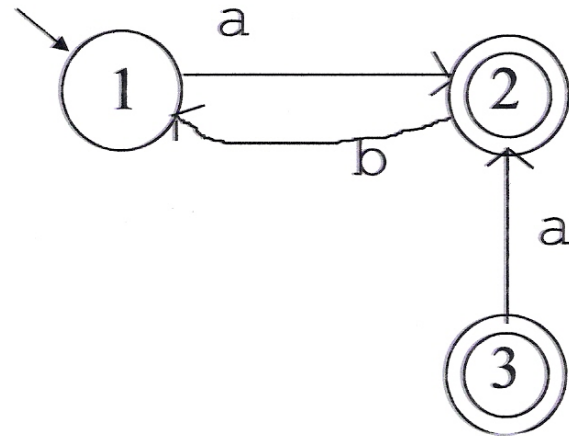
Consider:



Is this a minimal machine?

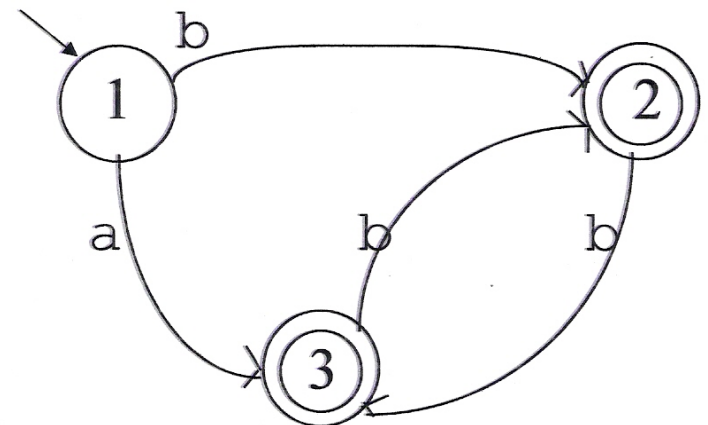
State Minimization

Step (1): Get rid of **unreachable** states.



State 3 is unreachable.

Step (2): Get rid of **redundant** states.

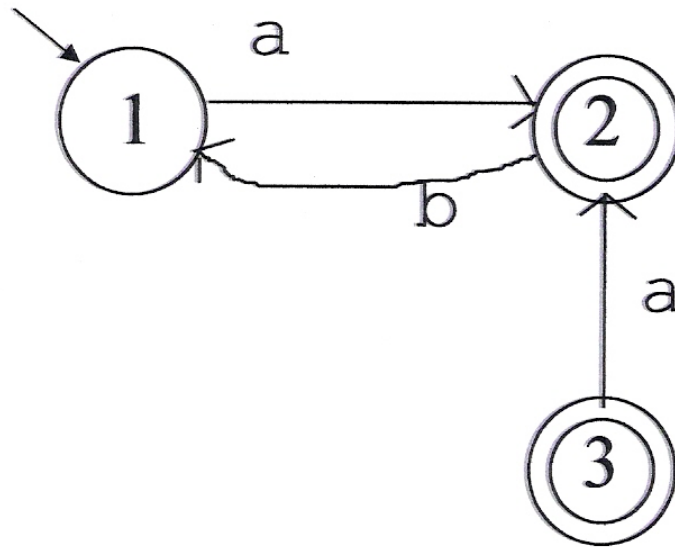


States 2 and 3 are redundant.

Getting Rid 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.

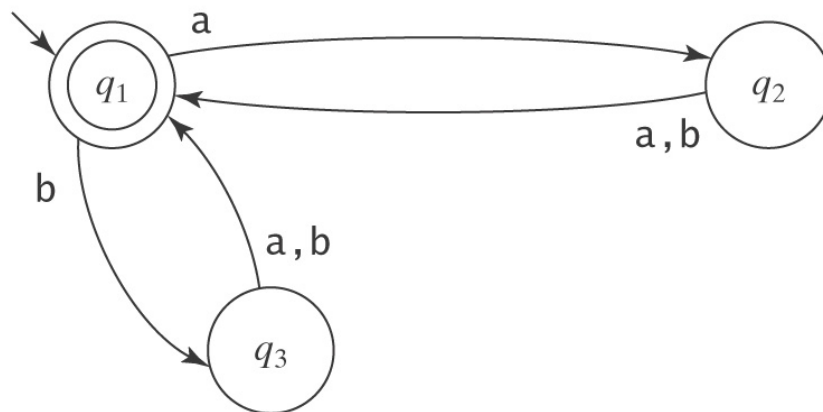
An algorithm for finding the reachable states:



Getting Rid of Redundant States

Intuitively, two states are **equivalent** to each other (and thus one is redundant) if all strings in Σ^* have the same fate, regardless of which of the two states the machine is in. But how can we tell this?

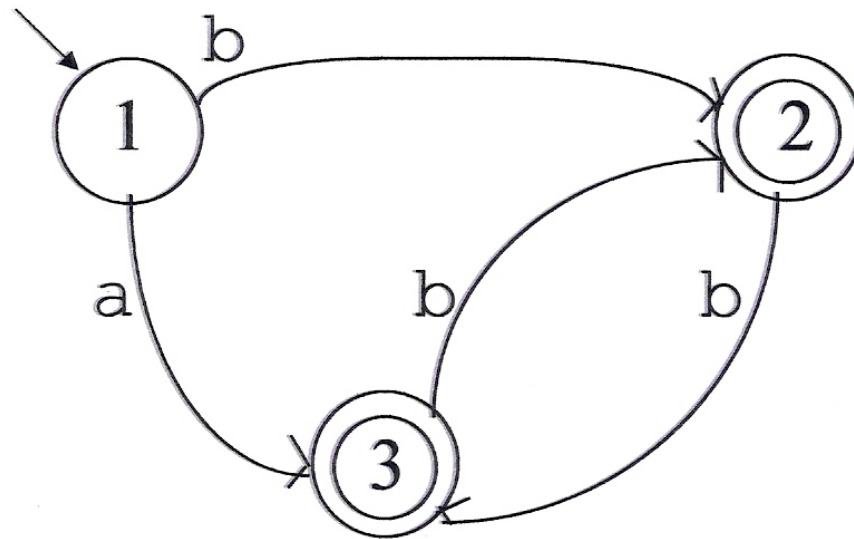
The simple case:



Two states have identical sets of transitions out.

Getting Rid of Redundant States

The harder case:



The outcomes in states 2 and 3 are the same, even though the states aren't.



Finding a Minimal DFSA

Two problems:

1. Given a regular language, find a minimal DFSA for it.
2. Given a DFSA, find a minimal DFSA equivalent to it.

We focus first on 1. Given a language:

- Capture the notion of equivalence classes of strings with respect to that 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 DFSA.

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

Equivalent, or **indistinguishable**, string with respect to L :

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

\approx_L is an Equivalence Relation

\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:

$$\forall x, y, z \in \Sigma^*$$

$$(((xz \in L \leftrightarrow yz \in L) \wedge (yz \in L \leftrightarrow wz \in L)) \rightarrow \\ (xz \in L \leftrightarrow wz \in L)).$$



\approx_L is an Equivalence Relation

Because \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 .



An example of \approx_L

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

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

The equivalence classes of \approx_L :

$[\varepsilon, aa, ab, ba, bb, aaaa, aaab, \dots]$ – even length

$[a, b, aaa, aab, aba, abb, \dots]$ – odd length

Another example

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

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

The equivalence classes of \approx_L :

$[\varepsilon, b, abb, \dots]$ [all strings in L].

$[a, abbba, \dots]$ [all strings that end in a and
have no prior a that is not
followed by a b].

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



Yet Another Example of \approx_L

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

$$L = aab^*a$$

The equivalence classes of \approx_L :



When More Than One Class Contains Strings in L

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

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

The equivalence classes of \approx_L :

- | | |
|-----|-----------------------------|
| [0] | $[\varepsilon]$ |
| [1] | $[a, aba, ababa, \dots]$ |
| [2] | $[b, ab, bab, abab, \dots]$ |
| [3] | $[aa, abaa, ababb\dots]$ |



Does \approx_L Always Have a Finite Number of Equivalence Classes?

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

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

ε	aa	aaaa
a	aba	aaaaa
b	aaa	

The equivalence classes of \approx_L :

The Best We Can Do

Theorem 5.4: Let L be a regular language and let M be a DFSA that accepts L . Then, 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 equivalence classes of \approx_L .

The Best Is Unique

Theorem 5.5: 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.

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 .

Proof, Continued

We must show that:

- **K is finite.** Since L is regular, it is accepted by some DFSA M' . M' has some finite number of states m . By Theorem 5.4 (see above), $n \leq m$. 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 .

Proof, Continued

- $L(M) = L$.

We prove first that, for any string w in Σ^* :

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

$$\begin{aligned} ([\varepsilon], w) &= ([\varepsilon], w_1 w_2 \dots w_m) & w_i \in \Sigma, 1 \leq i \leq m \\ &\vdash_M ([w_1], w_2 w_3 \dots w_m) \\ &\vdash_M ([w_1 w_2], w_3 w_4 \dots w_m) \\ &\vdash_M \dots \\ &\vdash_M ([w_1 w_2 \dots w_m], \varepsilon) = ([w], \varepsilon) \end{aligned}$$

M accepts w iff $[w] \in A = \{[x] : x \in L\}$ iff $w \in L$

So M accepts precisely the strings in L , that is, $L(M) = L$.

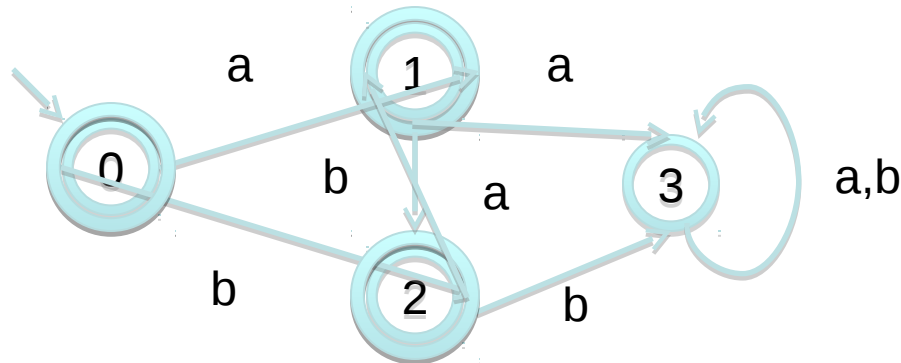
Proof, Continued

- There exists no smaller machine $M^\#$ that also accepts L . This follows directly from Theorem 5.4 (see above), which says that the number of equivalence classes of \approx_L imposes a lower bound on the number of states in any DFSA that accepts L .
- There is no different machine $M^\#$ that also has n states and that accepts L .

Example

- $L = \{w \in \{a,b\}^* : \text{no adjacent characters are the same}\}$
- equivalence classes of \approx_L :
- [0] $[\varepsilon]$ = start state
- [1] $[a, ba, aba, baba, \dots]$
- [2] $[b, ab, bab, abab, \dots]$
- [3] $[aa, bb, abaa, ababb\dots]$

$$\delta([x], a) = [xa]$$



The Myhill-Nerode Theorem

Theorem 5.6 (Myhill-Nerode): 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.

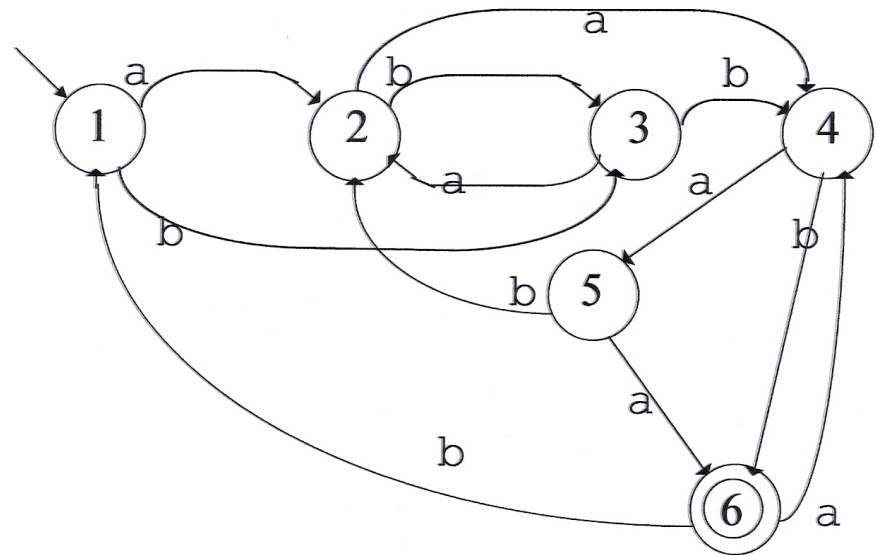
The number of equivalence classes of \approx_L is finite $\rightarrow L$ regular: If the cardinality of \approx_L is finite, then the construction that was described in the proof of the previous theorem will build an FSM that accepts L . So L must be regular.

So Where Do We Stand?

1. We know that for any regular language L there exists a minimal accepting machine M_L .
2. We know that $|K|$ of M_L equals the number of equivalence classes of \approx_L .
3. We know how to construct M_L from \approx_L .
4. We know that M_L is unique up to the naming of its states.

But is this good enough?

Consider:





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.



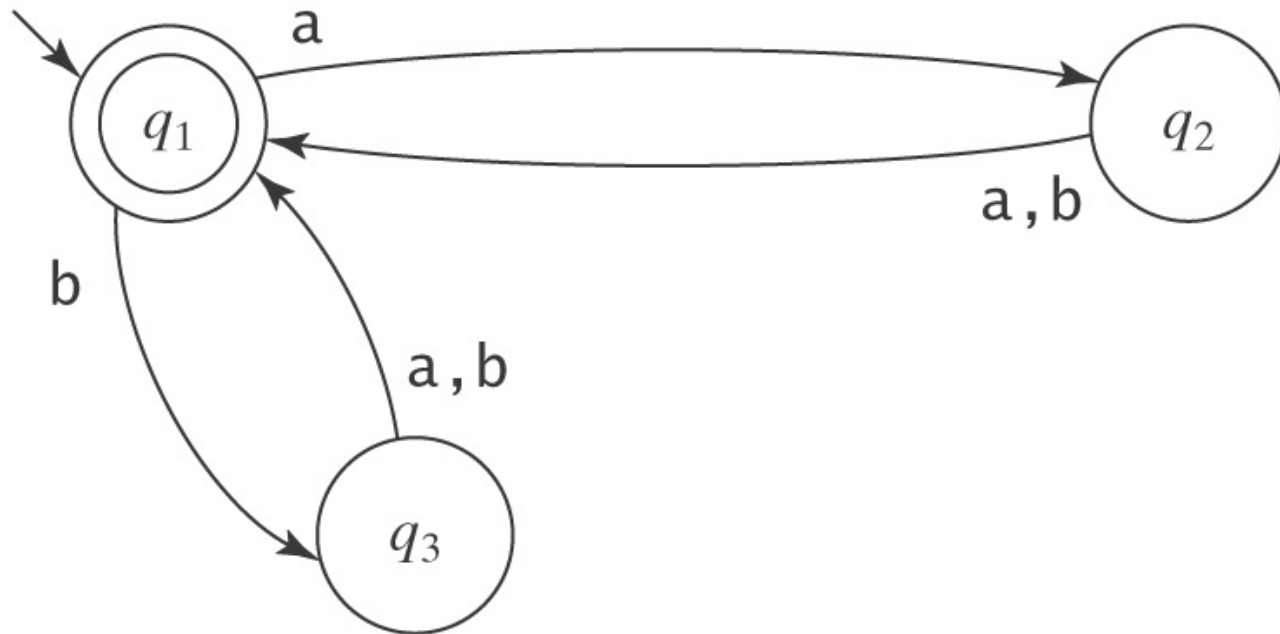
The Overclustering Approach

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 .

An Example

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



$$q_2 \equiv q_3$$



Constructing \equiv as the Limit of a Sequence of Approximating Equivalence Relations \equiv_n

(Where n is the length of the input strings that have been considered so far)

Consider input strings, starting with ε , and increasing in length by 1 at each iteration. Start by way overgrouping states. Then split them apart as it becomes apparent (with longer and longer strings) that their behavior is not identical.

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.



Constructing \equiv (cont'd)

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. $\forall a \in \Sigma (\delta(p, a) \equiv_{n-1} \delta(q, a))$

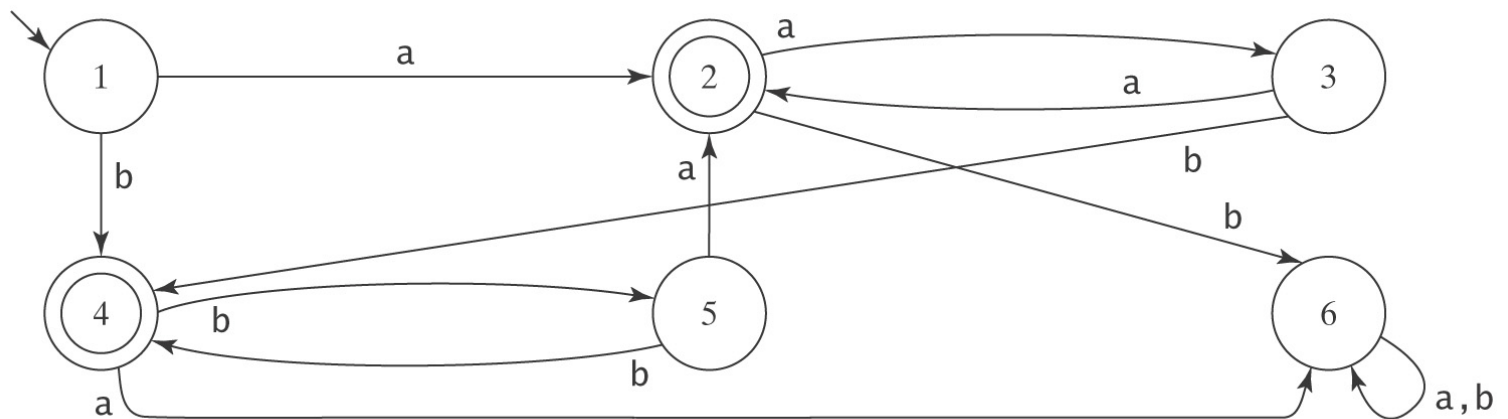
MinDFSM

MinDFSM(*M*: DFSM) =

1. *classes* := {*A*, *K-A*};
2. While $\exists E \in \textit{classes}, p, q \in E, c \in \Sigma$ with $[\delta(p, c)] \neq [\delta(q, c)]$ do
3. split *E* such that $[p] \neq [q]$
4. remove *E* from *classes*
5. add the subclasses of *E* to *classes*
6. return $M^* = (\textit{classes}, \Sigma, \delta, [s_M], \{[q]: q \in A_M\})$, where δ_{M^*} is constructed as follows:
 if $\delta_M(q, c) = p$, then $\delta_{M^*}([q], c) = [p]$

An Example

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

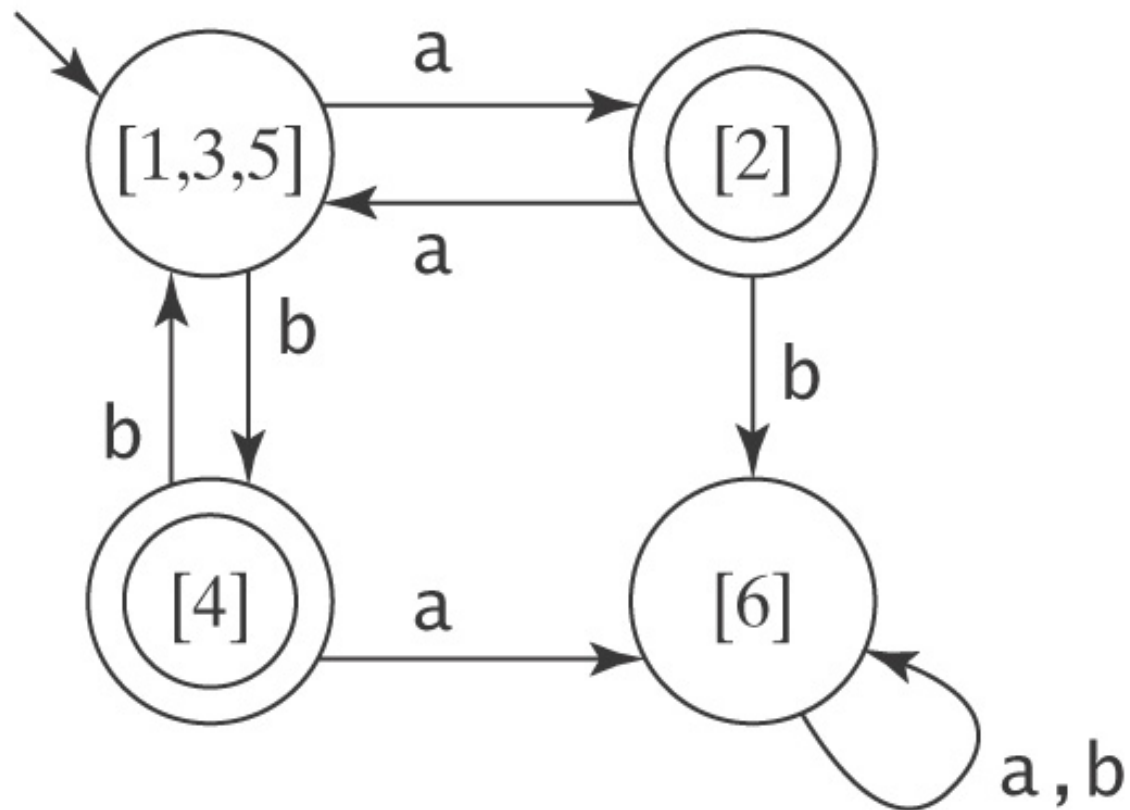


$$\equiv_0 = \{[1,3,5,6], [2,4]\}$$

$$\equiv_1 = \{[1,3,5], [6], [2,4]\}$$

$$\equiv_2 = \{[1,3,5], [2], [4], [6]\}$$

The Result



Summary

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