Computational Learning Theory

Learning Finite State Automata

Akihiro Yamamoto 山本 章博
http://www.iip.ist.i.kyoto-u.ac.jp/member/akihiro/
akihiro@i.kyoto-u.ac.jp

Learning Automata

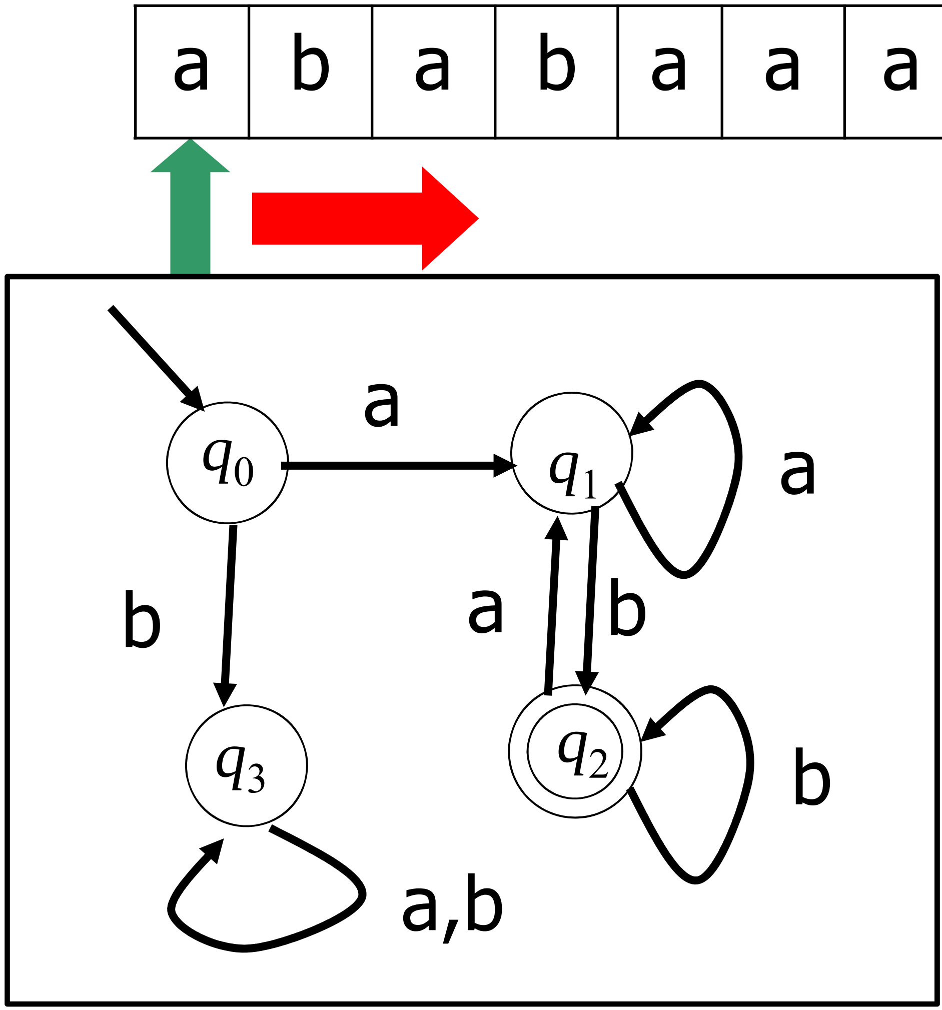
Learning Problems

- Find an FA which accepts the strings in C and rejects the strings in D.

  C = {ab, aab, abaab, aaab, aaaabbbb,abab}

  D = {a, b, bbbb, abba, baaaaba, babb}

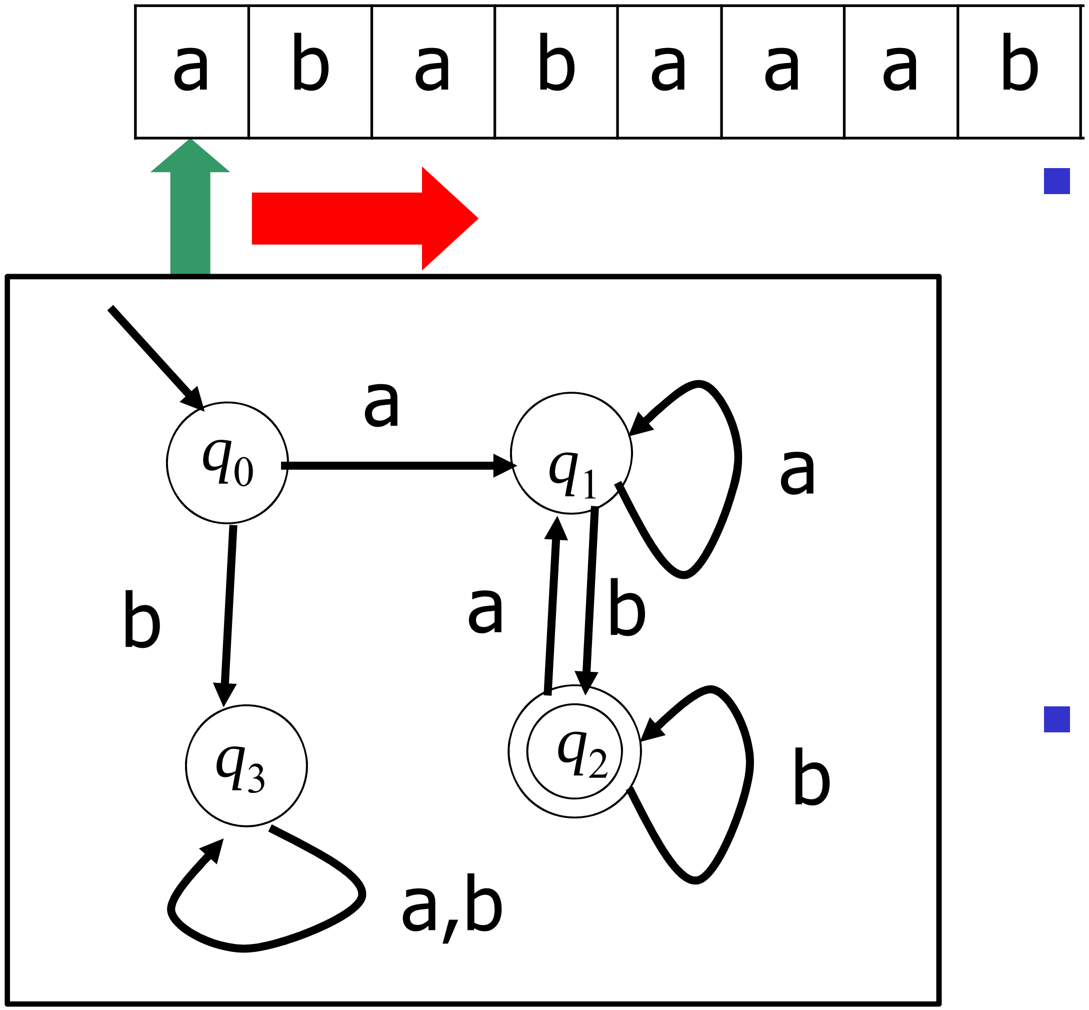
What is an Automaton?



- A change of observed squares left to right only, together with a possible change of state of mind.
- No change of symbols in the squares.

Machines of this type are called finite state automata.

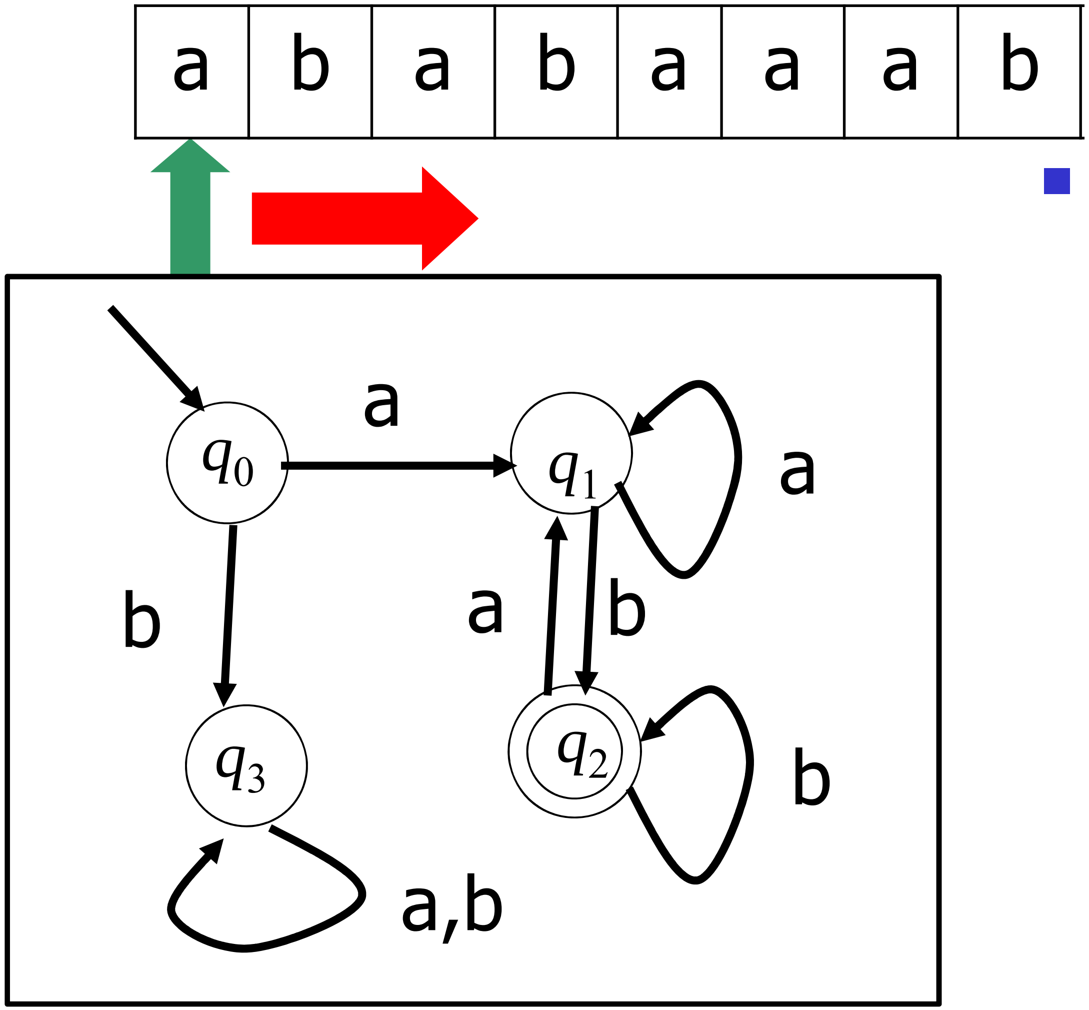
Distinguishing Strings with a FA



- The input string is accepted by the finite state automaton iff the transition ends at a finial state.
- The set of all strings accepted by the automaton is a formal language.

$L(M)$ = {aab, abb,aaab, aabb,abab, ,... }

Table Representation(1)



- The automaton is represented in the form of a table.

	F	a	b
q_0		q_1	q_3
q_1		q_1	q_2
q_2	v	q_1	q_2
q_3		q_3	q_3

Table Representation(2)

- Mathematically, a finite state automaton is represented in the form M=(Σ, S, δ, s_0, F)

  where

  Σ is the alphabet,

  S is a set of states,

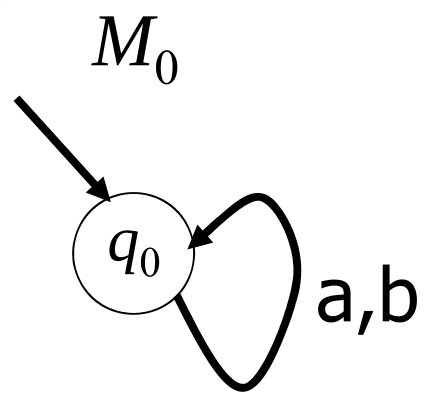
  $\delta : S \times \Sigma \rightarrow S$ is a transition function represented as a transition table,

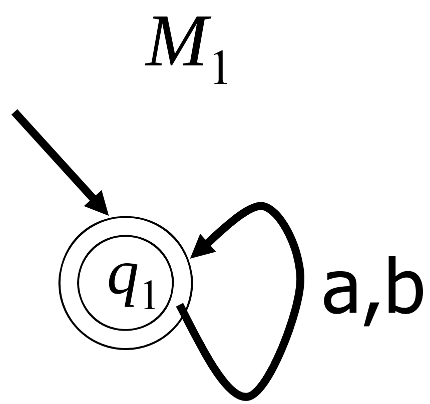
  $q_0 \in S$ is an initial state,

  $F \subset S$ is a set of final states.

	F	a_1	…	a_n
q_0				
$\cdots$				
q_m				

Finite Automata of One State





	F	a	b
q_0		q_0	q_0

	F	a	b
q_0	v	q_0	q_0

$$L(M_0) = \varnothing$$

$$L(M_0) = \Sigma^*$$

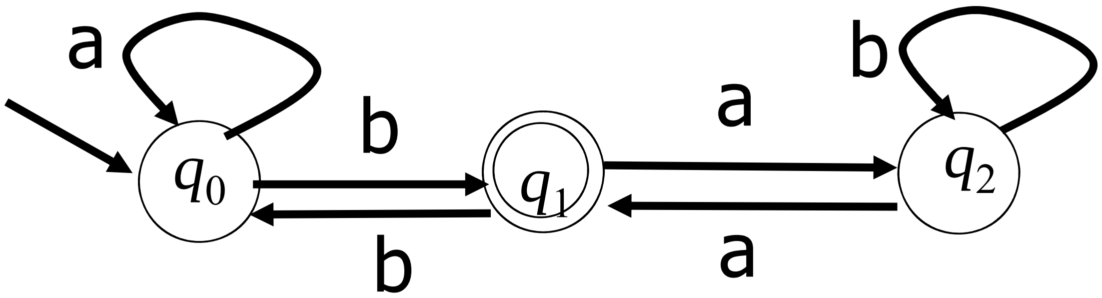
Equivalence of REs and FAs

Theorem [McNorton-Yamada]

- Every regular expression R can be transformed into a finite state automaton so that $L(R) = L(M)$.
- Every finite state automaton M can be transformed into a regular expression R so that $L(M) = L(R)$.

McNorton-Yamada's Method(1)

- Step –1. Let $\boldsymbol{R_{ij}}$ is the regular expression representing the set of symbols which directly transits from q_i to q_j . If a transition from q_i to itself exists, add ε to the set.



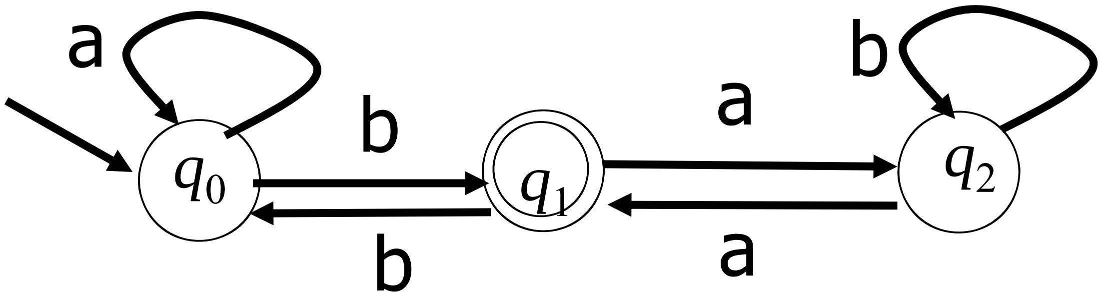
$$\boldsymbol{R_{00}} = \mathbf{a} + \varepsilon,\ \ \boldsymbol{R_{11}} = \varnothing + \varepsilon = \varepsilon,\ \boldsymbol{R_{22}} = \mathbf{b} + \varepsilon,$$

$$\boldsymbol{R_{01}} = \mathbf{b},\ \ \boldsymbol{R_{02}} = \varnothing,\ \boldsymbol{R_{10}} = \mathbf{b},\ \boldsymbol{R_{12}} = \mathbf{a},$$

$$\boldsymbol{R_{20}} = \varnothing,\ \boldsymbol{R_{21}} = \mathbf{a},$$

McNorton-Yamada's Method(2)

- Step 0a. Let R^0_{ii} is the regular expression representing the set of strings which transits from q_i to itself directly or via q_0.



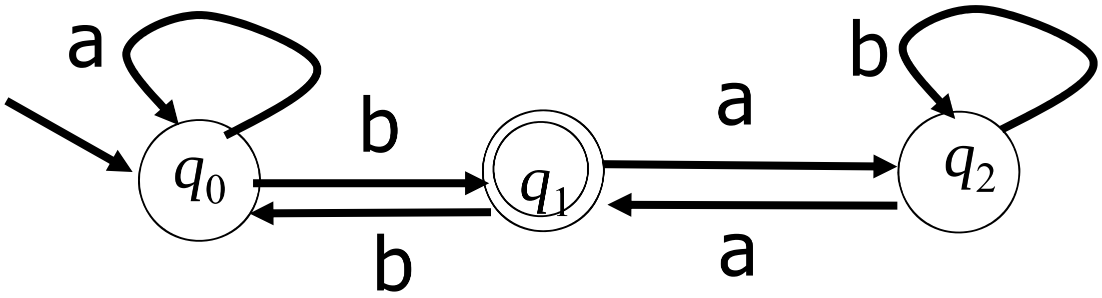
$$R^0_{11} = R_{11} + R_{10}(R_{00})^* R_{01}$$

$$= (\varepsilon) + (\mathbf{b} + \varepsilon)\,(\mathbf{a} + \varepsilon)^*\,(\mathbf{b} + \varepsilon)$$

$$= \varepsilon + \mathbf{b}\,\mathbf{a}^*\,\mathbf{b}$$

McNorton-Yamada's Method(3)

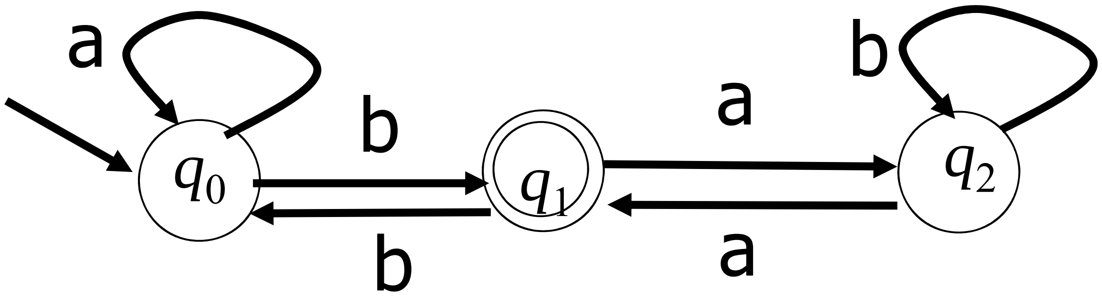
- Step 0b. Let R^0_{ij} is the regular expression representing the set of strings which transits from q_i to q_j directly or via q_0.



$$R^0_{12} = R_{12} + R_{10}(R_{00})^* R_{02}$$
$$= \mathbf{a} + (\mathbf{b} + \varepsilon)\,(\mathbf{a} + \varepsilon)^* \varnothing$$
$$= \mathbf{a}$$

McNorton-Yamada's Method(4)

- Step 1. Let $\boldsymbol{R}^{1}_{ij}$ is the regular expression representing the set of strings which transits from q_i to q_j directly, or via q_0 or q_1.



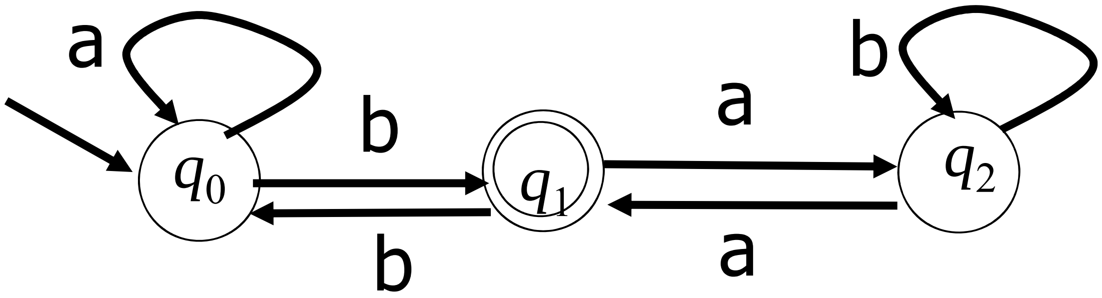
$$R^{1}_{02} = R^{0}_{02} + R^{0}_{01}(R^{0}_{11})^{*}\, R^{0}_{12} =$$

$$= \varnothing + (\mathbf{a}^{*}\ \mathbf{b})\ (\varepsilon + \mathbf{b}\ \mathbf{a}^{*}\ \mathbf{b})^{*}\ (\mathbf{a})$$

$$= \mathbf{b} + \mathbf{a}^{*}\ \mathbf{ba} + \varepsilon + \mathbf{a}^{*}\ \mathbf{b}\ (\mathbf{b}\ \mathbf{a}^{*}\ \mathbf{b})^{*}\ \mathbf{a}$$

McNorton-Yamada's Method(4)

- Step 2. Let R^2_{ij} is the regular expression representing the set of strings which transits from q_i to q_j directly, or via q_0 or q_1 or q_2.

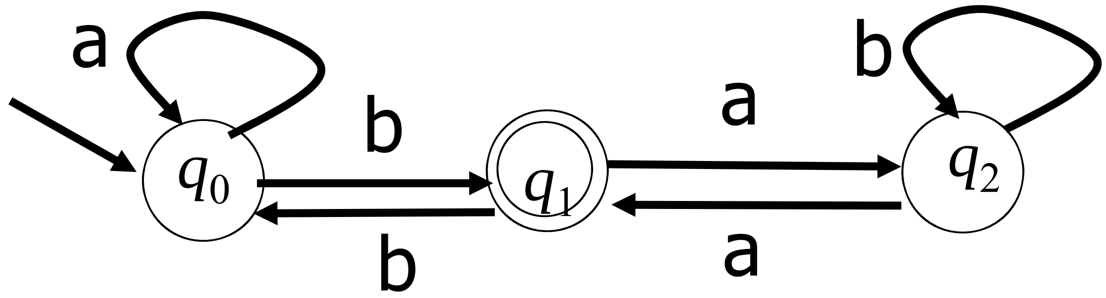


$$R^2_{02} = R^1_{02} + R^1_{01}(R^1_{11})^* R^1_{12}$$

McNorton-Yamada's Method(6)

- Step k. Let R^k_{ij} is the regular expression representing the set of strings which transits from q_i to q_j

  directly, or via q_0 or q_1 or … or q_k.



$$R^2_{02} = R^1_{02} + R^1_{01}(R^1_{11})^* R^1_{12}$$

Formulation of Learning FA

- Formulation of Learning

$$\mathrm{argmin}_{M \in \mathsf{FA}} \left(\Sigma_{\boldsymbol{x} \in Data} \mathrm{Loss}(M, \boldsymbol{x}) + \lambda\, \mathrm{P}(M)\right)$$

where FA : the set of all finite state automata,

Data : a finite set of pairs $\boldsymbol{x} = <w, s>$ of a string with a sign such that $s = +$ if $w \in C$ and $s = -$ if $w \in D$,

$$\mathrm{Loss}(M, \boldsymbol{x}) = \begin{cases} 0 \ \text{ if } \ \boldsymbol{x} = <w, +> \text{ and } w \in L(M) \\ \quad\quad \text{ or } \boldsymbol{x} = <w, -> \text{ and } w \notin L(fM), \\ \infty, \text{ otherwise,} \end{cases}$$

P(M) : the number of states in M

A Simple Generate-and-Test Algorithm

- Assume we have a method to generate a new automaton.

Let the input data $x_1, x_2, \ldots, x_N$
Initialize M as some automaton.
for $k = 1,2,\ldots$
  $M_k = M_{k-1}$
  for $n = 1,2,\ldots, N,$
    if $(x_n \in C$ and $x_n \notin L(M_k))$ or $(x_n \in D$ and $x_n \in L(M_k))$
      replace M_k with another M'
  if $M_k = M_{k-1}$
    terminate and output M_k

- With which M' should we replace M?

Simple Strategy of Learning

- With referring the existence of minimum FA, we can easily imagine a simple strategy of learning:

  Generate all FA, and enumerate them from small to large according to their sizes.

Representation of Finite State Automata

- Mathematically, a finite state automaton is represented in the form $M=(\Sigma, S, \delta, s_0, F)$

  where

  Σ is the alphabet,

  S is a set of states,

  $\delta : S \times \Sigma \rightarrow S$ is a transition function represented as a transition table,

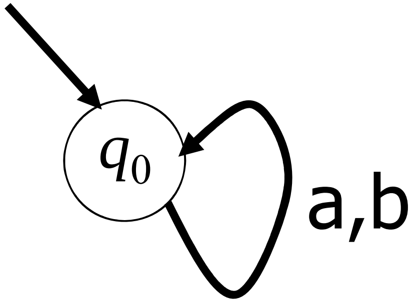
  $q_0 \in S$ is an initial state,

  $F \subset S$ is a set of final states.

	F	a_1	$\ldots$	a_n
q_0				
$\cdots$				
q_m				

Finite Automata of One State

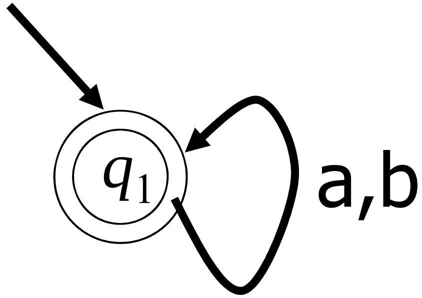
M_0



	F	a	b
q_0		q_0	q_0

$$L(M_0) = \varnothing$$

M_1



	F	a	b
q_0	√	q_0	q_0

$$L(M_0) = \Sigma^*$$

Generation by Enumeration

- We can make an infinite but effective enumeration of all automata, because

  every automaton can be represented

  as a transition table.

  - This means that we can have an infinite sequence of automata

    $M_1, M_2, \ldots$

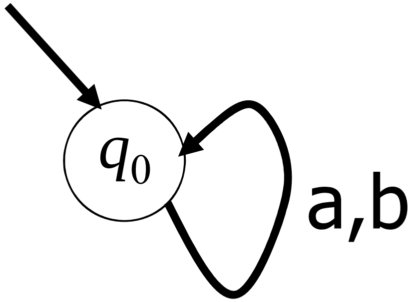
    any automaton M appears as $M_i = M$.

	F	a_1	$\ldots$	a_n
q_0				
$\ldots$				
q_m				

- In the algorithm $M = M_i$ is just replaced with $M' = M_{i+1}$.

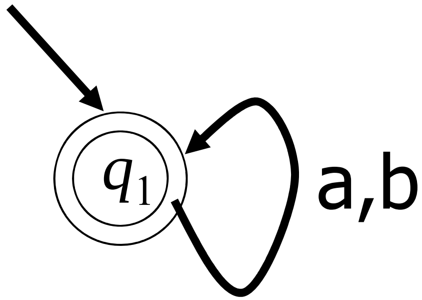
Enumeration of Automata(1)

M_0



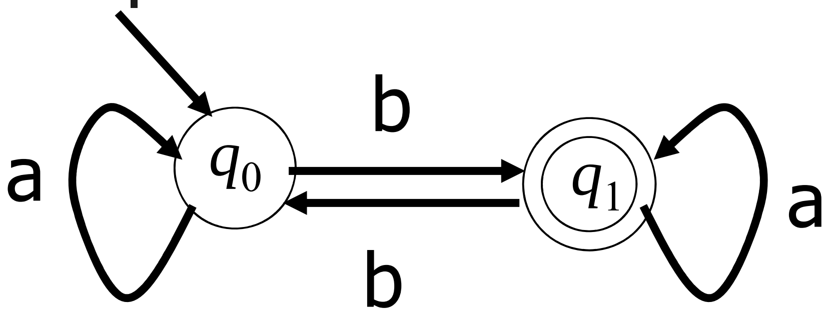
	F	a	b
q_0		q_0	q_0

M_1



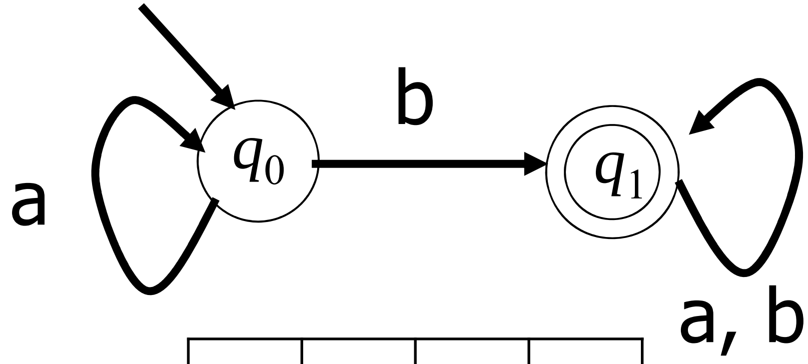
	F	a	b
q_0	v	q_0	q_0

Enumeration of Automata(2)



	F	a	b
q_0		q_0	q_1
q_1	v	q_1	q_0

	F	a	b
q_0	v	q_0	q_1
q_1		q_1	q_0



	F	a	b
q_0		q_0	q_1
q_1	v	q_1	q_1

	F	a	b
q_0	v	q_0	q_1
q_1		q_1	q_1

…

A Simple Generate-and-Test Algorithm

Assume a procedure of enumerating all FA so that the enumeration $M_0, M_1, M_2, \ldots, M_i, \ldots$ satisfies

$$P(M_0) \leq P(M_1) \leq P(M_2) \leq \ldots \leq P(M_i) \leq \ldots$$

Let the input data $x_1, x_2, \ldots, x_N$
Initialize $M = M_0$ as an automaton consisting of one state
let $k = 0$
forever
    let $k' = k$
    for $n = 1,2,\ldots, N,$
        if $(x_n \in C$ and $x_n \notin L(M_{k'}))$ or $(x_n \in D$ and $x_n \in L(M_{k'}))$
            replace k with $k + 1$
    if $k' = k$
        terminate and output M_k

Some Properties of the Algorithm

- The algorithm always terminates because
  for any pair of C and D ($C \cap D = \emptyset$),
  there exists a finite state automaton M such that
  $L(M) = C$ and $L(M) \cap D = \emptyset$, and this M
  appears in the enumeration as $M_i = M$.
- If the enumeration is made so that "smaller automata appear earlier", the algorithm returns the smallest automaton M such that

$$L(M) \subset C \text{ and } L(M) \cap D = \emptyset.$$

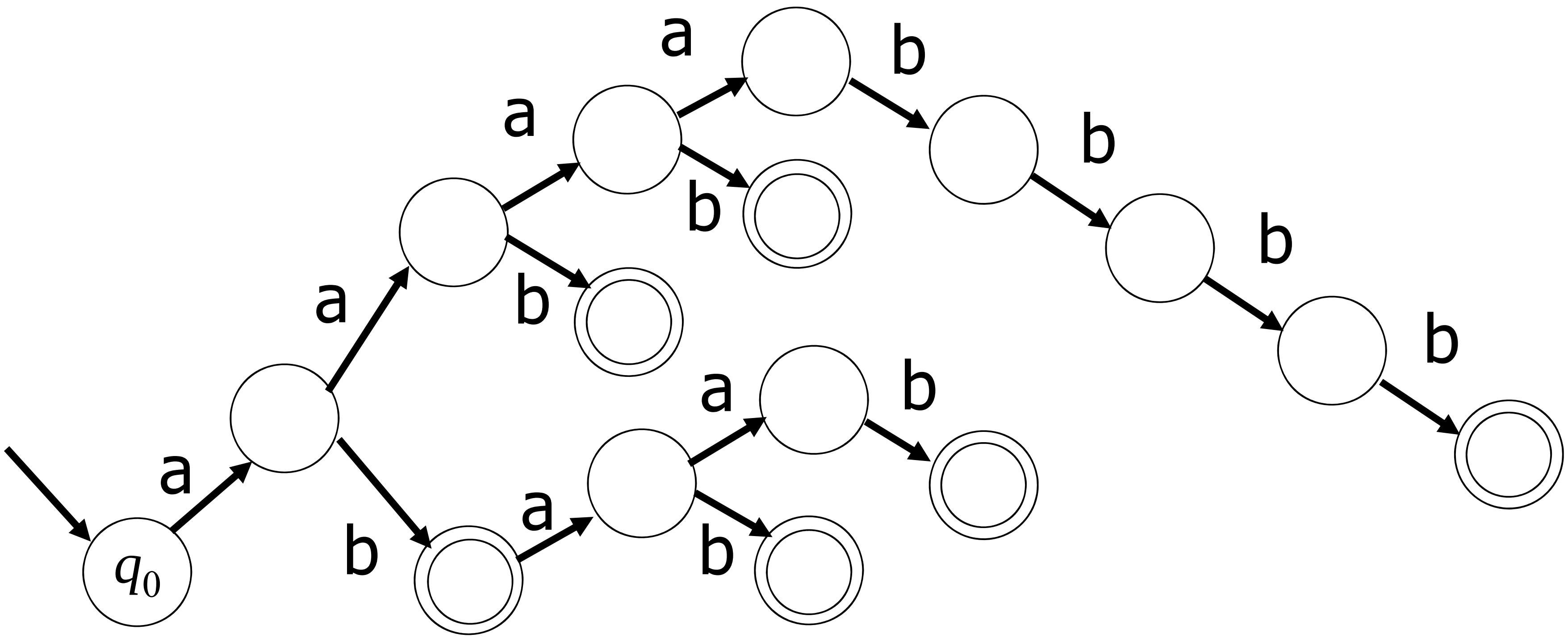
Note 1

- There might be several automata consistent with given C and D.
- For any finite set $C \subset \Sigma^*$, we can easily construct a finite state automaton which accepts only the strings in C, and rejects all strings not contained in C.
  - The FA is called a prefix tree automaton.

Example

C_1 = {ab, aab, abaab, aaab, aaaabbbb,abab}

D_1 = {a, b, bbbb, abba, baaaaba, babb}



Prefixes of a String

Definition A string $u \in \Sigma^*$ is a prefix of another string $s \in \Sigma^*$

$\Leftrightarrow$ There exists a string $v \in \Sigma^*$ such that $s = uv$.

For a set $S \subseteq \Sigma^*$, we let

$$P(S) = \{ u \in \Sigma^* \mid u \text{ is a prefix of some } s \text{ in } S \}.$$

Example The prefixes of aab are ε, a, aa, and aab,

the prefixes of ab are ε, a, and ab, and so we have

$$P(\{ab , aab\}) = \{\varepsilon, a, aa, ab , aab\}.$$

Prefix Tree Automata

Definition A prefix tree automaton of a finite set $S \subseteq \Sigma^*$ is defined as

$$M = (\Sigma, Q = Q_{\mathrm{P}(S)}, \delta, q_0 = q_\varepsilon, F = Q_S)$$

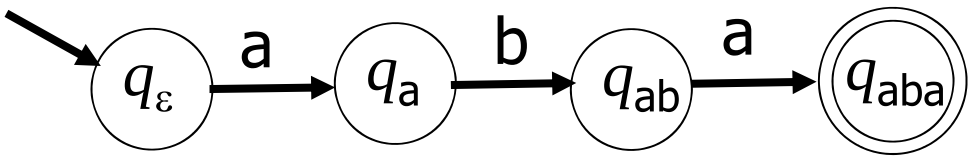
where

$$Q_{\mathrm{P}(S)} = \{ q_s \mid s \in \mathrm{P}(S) \},$$

$$\delta(q_s, c) = q_{sc} \quad \text{if } s \in \mathrm{P}(S) \text{ and } sc \in \mathrm{P}(S),$$

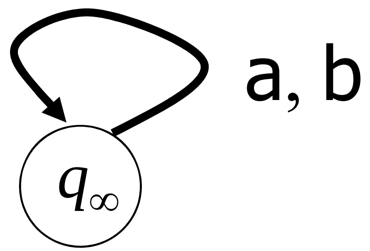
$$Q_S = \{ q_s \mid s \in S \}$$

$s =$ aba



Note

- The automaton does not satisfy the mathematical definition because, for example, no transition from q_0 is defined for the symbol b.
  - This means that δ is not a mathematical function, but a partial function.
- This fault can be easily recovered by adding a special state q_∞ (called a dead state) and letting every missing value of δ be q_∞.
- Under assuming this recover, we modify the definition.



Finite state automata (3)

- A finite state automata is defined as

  $M = (\Sigma, Q, \delta, q_0, F)$

  where

  Q is a set of states

  $\delta : Q \times \Sigma \rightarrow Q$ is a partial transition function

  represented as a transition table

  $q_0 \in Q$ is an initial state

  $F \subset Q$ is a set of final state

Is the automaton pleasant?

- The prefix tree automaton *T* overfits *C*.
  - It accepts no strings which is not in *C*.
  - It must be revised if new examples are added to *C*.
    - It is a natural to assume that positive examples and negative are added more experiments or observations are made.
- The prefix tree automaton *T* does not generalize *C*.
  - Intuitively learning should be activity of making general guesses from examples.
  - The prefix automaton tree overgeneralize the set *D* of negative examples.

Note 2

- There is a minimum one in the sense that the number of states in it is minimum.
- Unfortunately it is proved that the problem of finding a minimum automaton consistent with given C and D is NP-hard.
  - The activity of a learning algorithm should not be evaluated (justified) only on the viewpoint of optimization.
  - Even though it were not ensured that the algorithm returns the best solution, the algorithm could work as “learning”.

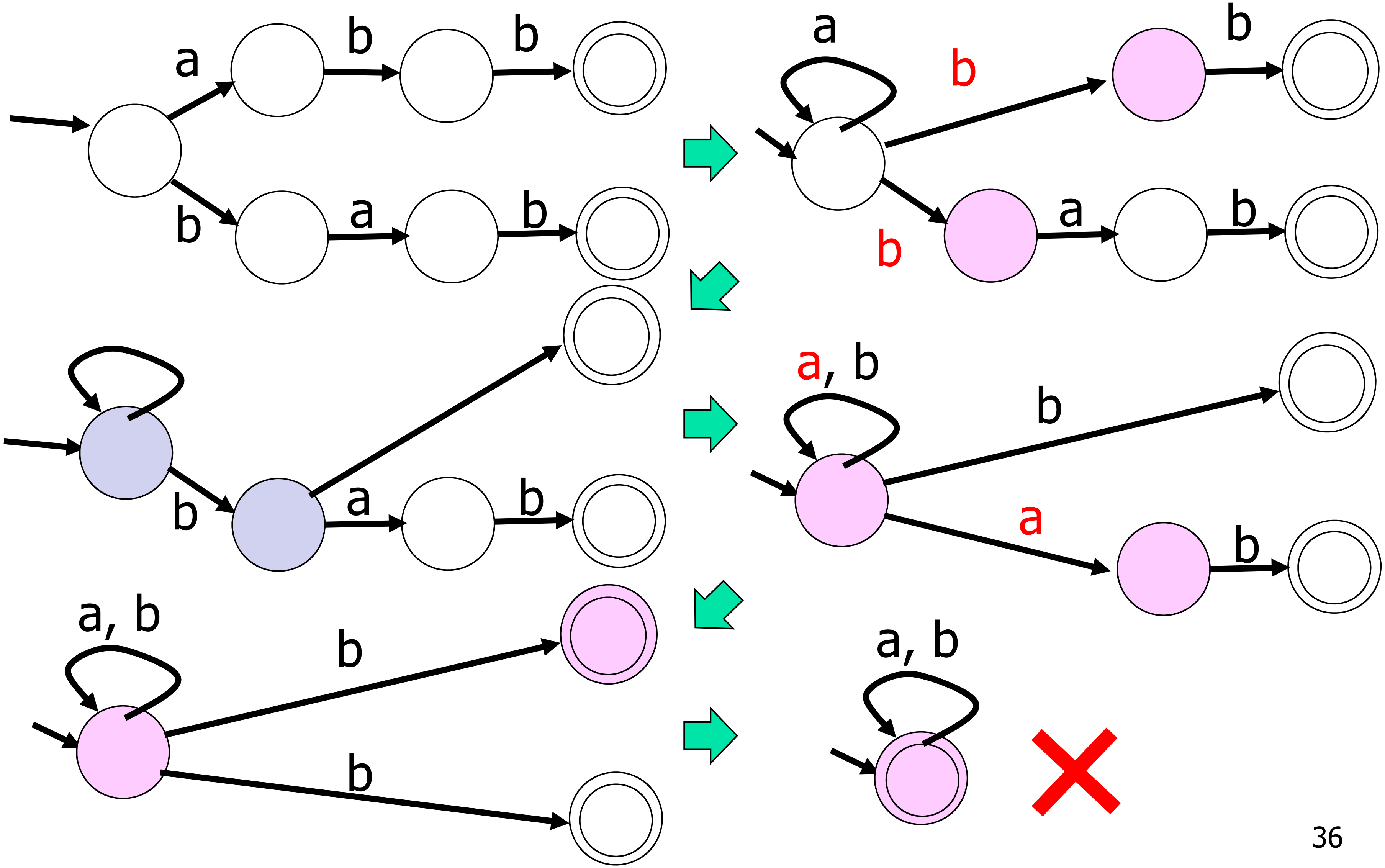
Generalization by Merging States

Generalization by Merging States

- The prefix tree T can be transformed into a more general automaton by merging several states into one states.

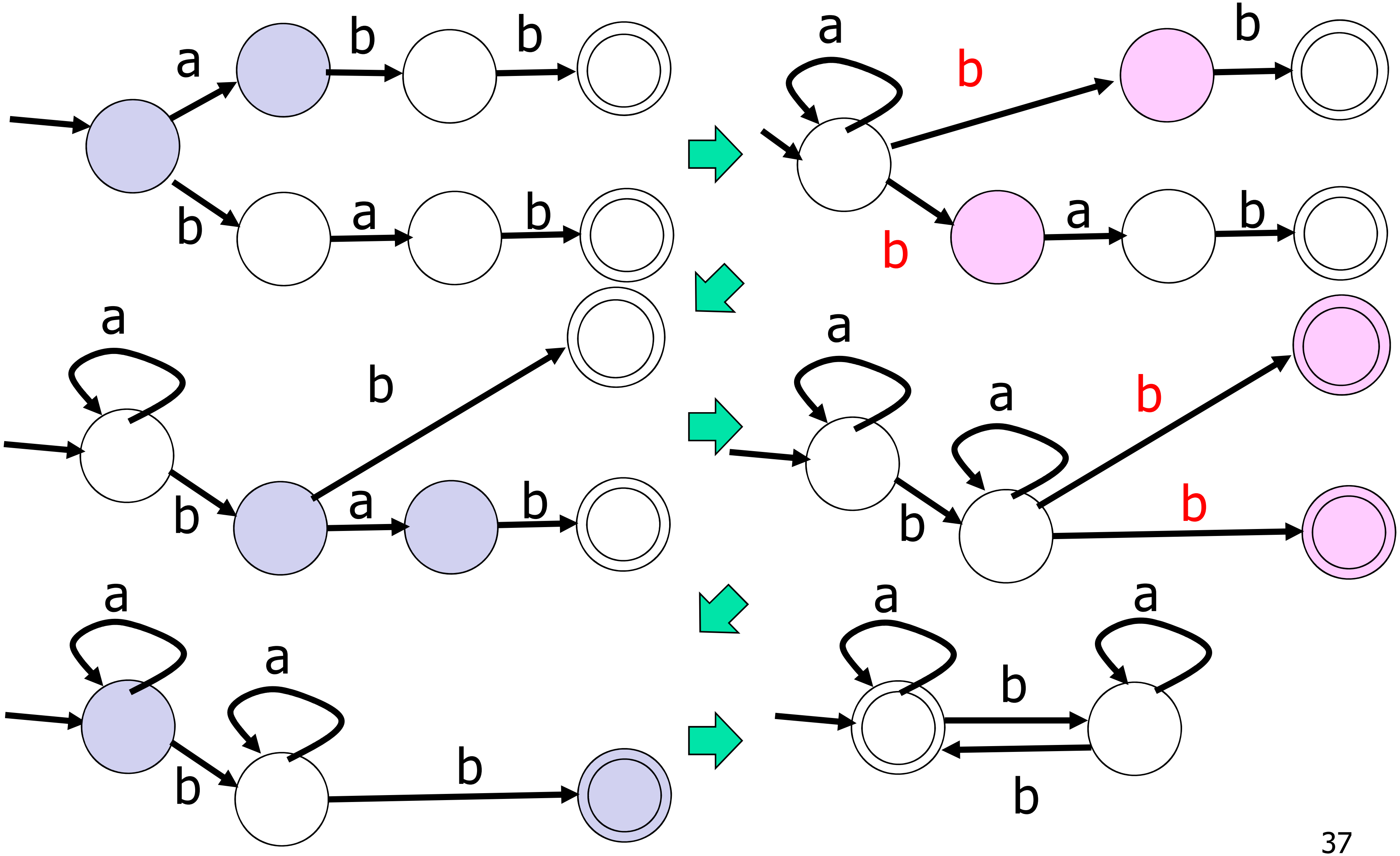
Example: Merging States

$C = \{\text{abb, bab}\}, D = \{\text{ab}\}$



Example: Merging States(cont.)

$C = \{\text{abb}, \text{bab}\}, D = \{\text{ab}\}$

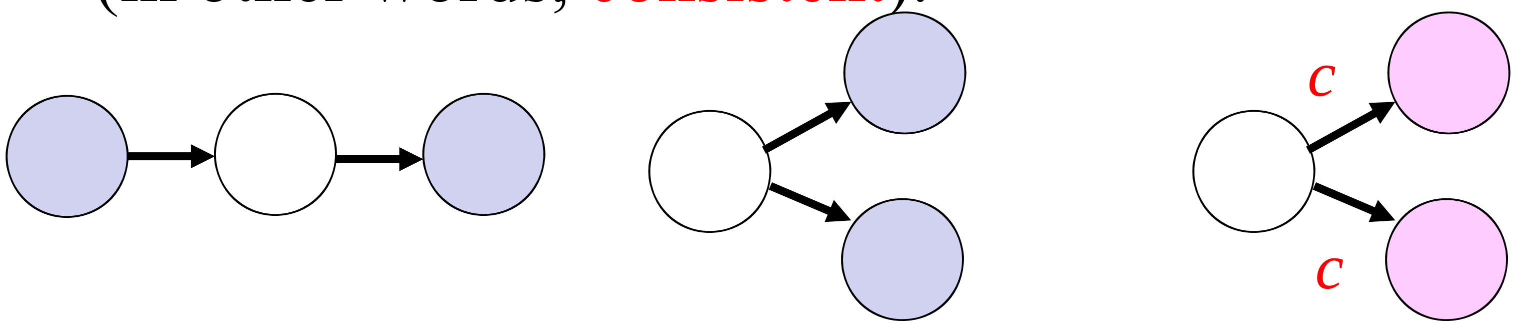


Two Types of Merge

- We have to treat two types of merge:

  1. Merging two states to generate a more general automaton, and

  2. Merging two states to keep the automaton deterministic (in other words, consistent).



- Strategy: first apply the first merge, and then try the second merge as far as possible.

Partitions and Blocks

Definition A partition of a set Q of states of a automaton, is a collection $\pi = \{B_1, B_2, \ldots, B_n\}$ of subsets of Q satisfying

1. every B_i is not empty,
2. $B_i \cap B_j = \varnothing$ for every pair of i and j such that $i \neq j$,
3. $B_1 \cup B_2 \cup \ldots \cup B_n = \varnothing$.

Every B_i is called a bock of π .

- A block $B = \{q_1, q_2, \ldots, q_m\}$ represents a state obtained by merging the states $q_1, q_2, \ldots, q_m$ into one.

Definition Let $\pi = \{B_1, B_2, \ldots, B_n\}$ be a partition of states. To merge two blocks B_i and B_j means to revise π to $\pi_{(i,j)} = \{B_1, B_2, \ldots, B_n\} - \{B_i, B_j\} \cup \{B_i \cup B_j\}$.

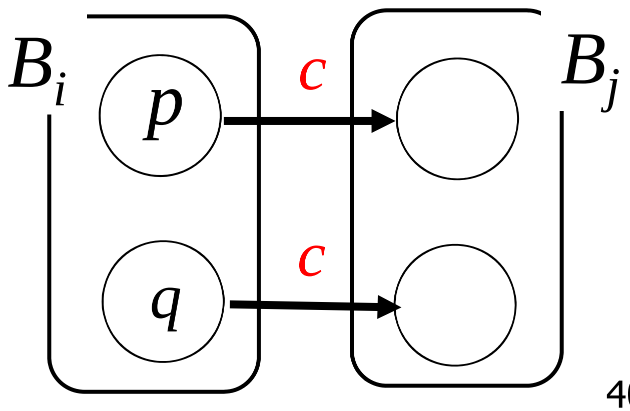
Consistent Partition

Definition A partition $\pi = \{B_1, B_2, \ldots, B_n\}$ for $M = (\Sigma, Q, \delta, q_0, F)$ is consistent

$\Leftrightarrow$

for every block B_i , every pair $p, q \in B_i$ and every symbol $c \in \Sigma$,

if both $\delta(p, c)$ and $\delta(q, c)$ are defined, then there is a block B_j such that both $\delta(p, c)$ and $\delta(q, c) \in B_j$.



Partitioned Automata

If a partition $\pi = \{B_1, B_2, \ldots, B_n\}$ for $M = (\Sigma, Q, \delta, q_0, F)$ is consistent we can define a partial function

$$\delta' : \pi \times \Sigma \to \pi$$

and also an automaton $M' = (\Sigma, \pi, \delta', B_0, F')$ with

$$F' = \{B_i \mid \text{some } q \in B_i \text{ is in } F\}.$$

The automaton is denoted M/π .

RPNI Algorithm[Oncina and Gracia92]

Regular Positive Negative Inference (PRNI) Algorithm

Inputs : $C \subset \Sigma^*$: a finite set of positive examples

$D \subset \Sigma^*$: a finite set of negative examples

Method : Make a list $[s_1, s_2, \ldots, s_n]$ of elements in $\mathrm{P}(C)$

Make the prefix automaton M of C; $k = 0$; $\pi_0 = \{\{q_s\} | s \in \mathrm{P}(C)\}$

for $i = 2$ **to** n

for $j = 1$ **to** $i - 1$

if $q_{si} \in B_i$ and $q_{sj} \in B_j$ such that $B_i \neq B_j$

let π' be the partition obtained by merging B_i and B_j

while π' is not consistent

Choose a pair $q' \in B'$ and $q'' \in B''$ violating the consistency

$\pi' :=$ the partition obtained by merging B' and B'' in π'

if M/π' rejects all strings in D

$\pi_k := \pi'$; $k := k + 1$

Output M/π_k

How to make the list of examples

- We have to fix a method of making the list $[s_1, s_2,\ldots,s_n]$ of $P(C)$.
- We had better use some order $<$ and make the list so that

$$s_1 < s_2 < \ldots < s_n$$

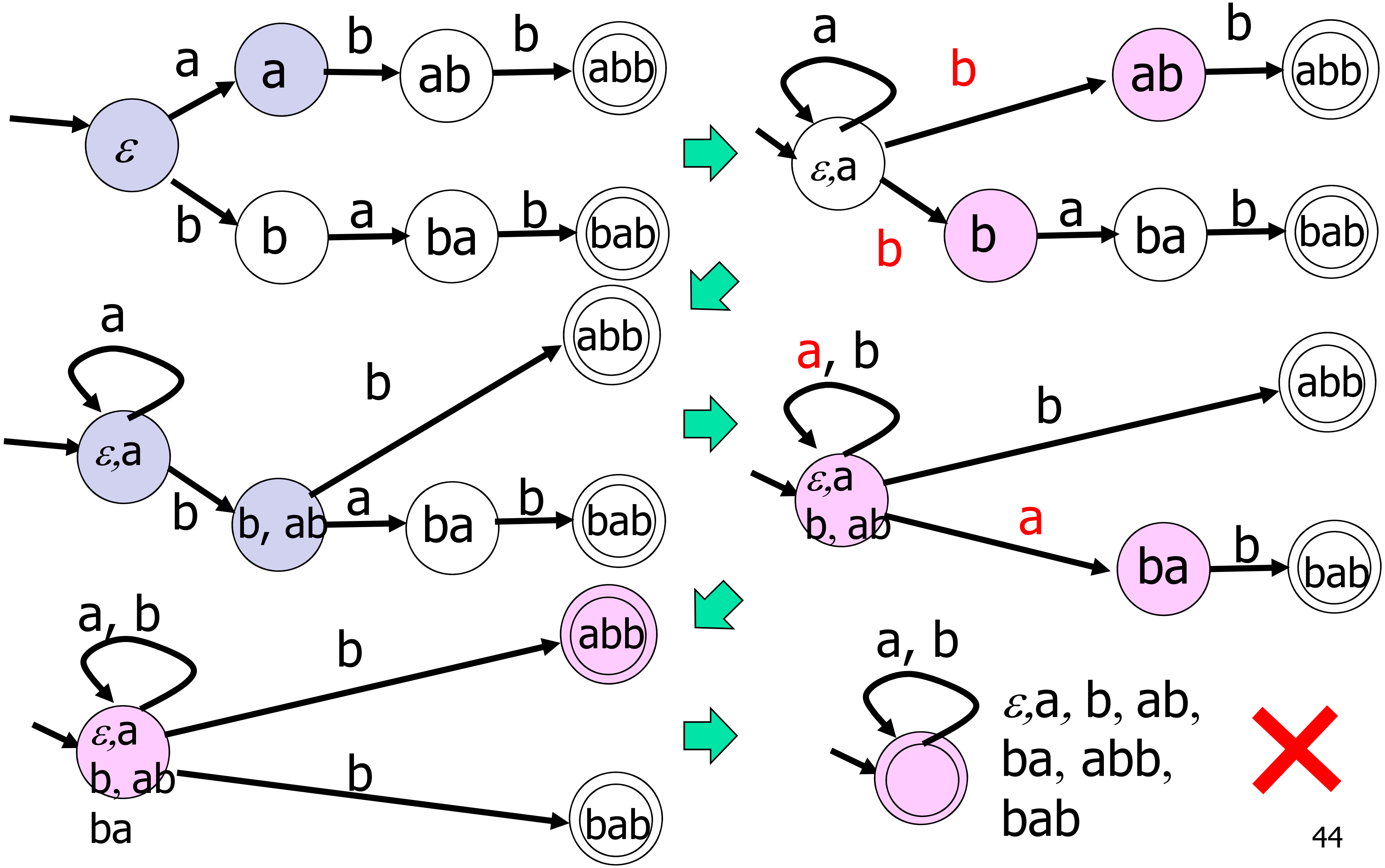
- We use the length-wise lexico-graphic order:

$$s < t \text{ if } |s| < |t| \text{ or}$$
$$|s| = |t| \text{ and } s \text{ is earlier than } t \text{ in the lexico-graphic order}$$

Example a < b < ab < ba < abb < bab

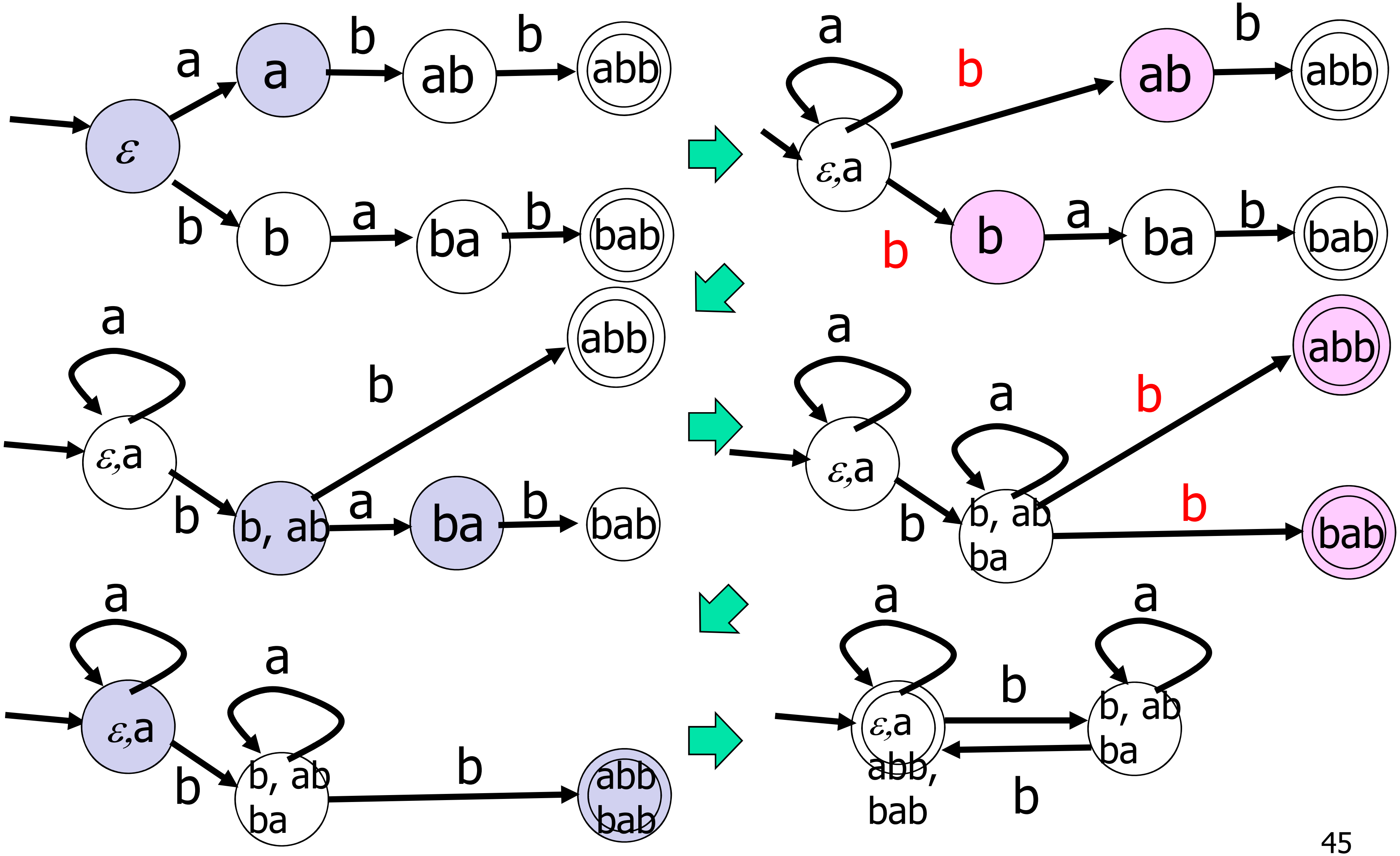
Example: Merging States

$C = \{\text{abb, bab}\}, D = \{\text{ab}\}$



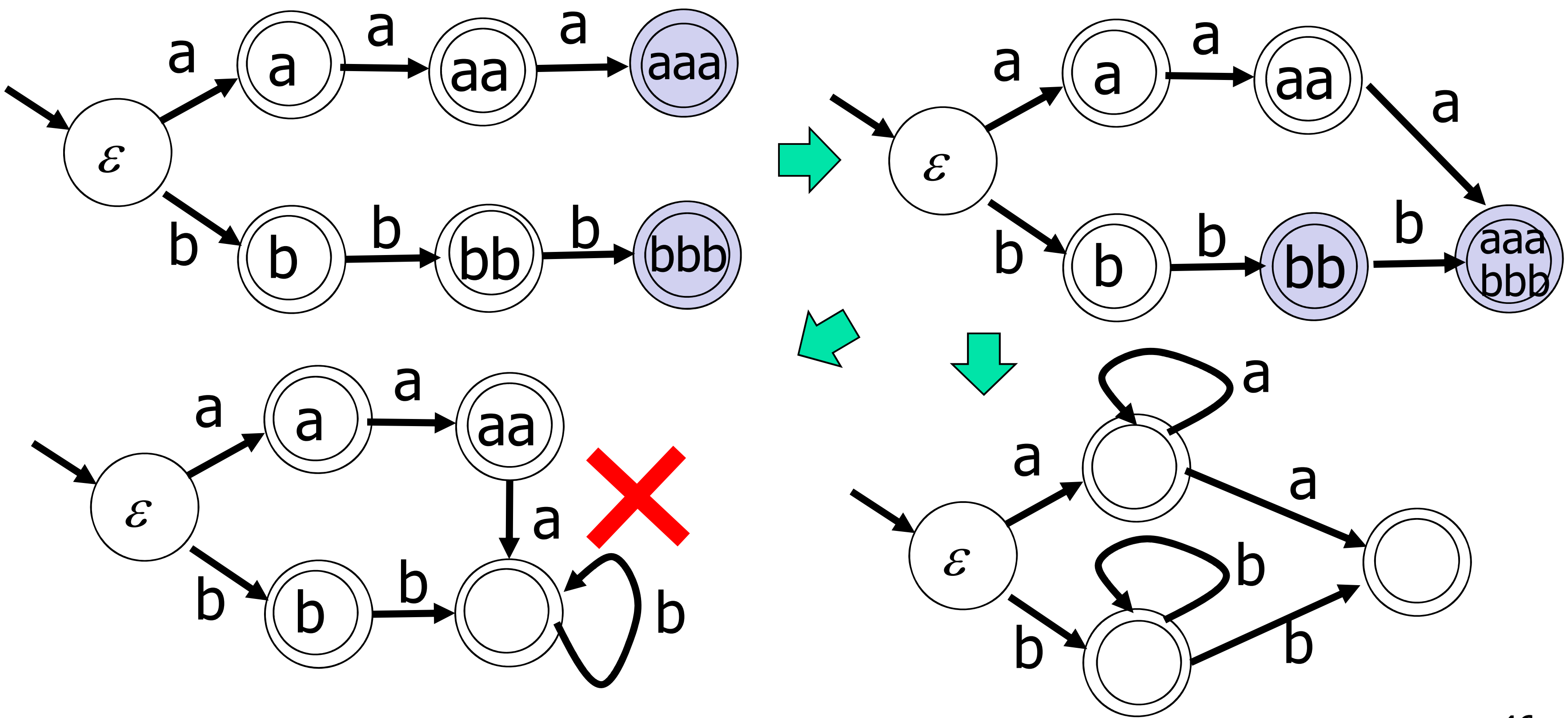
Example: Merging States(cont.)

$C = \{\text{abb, bab}\}, D = \{\text{ab}\}$



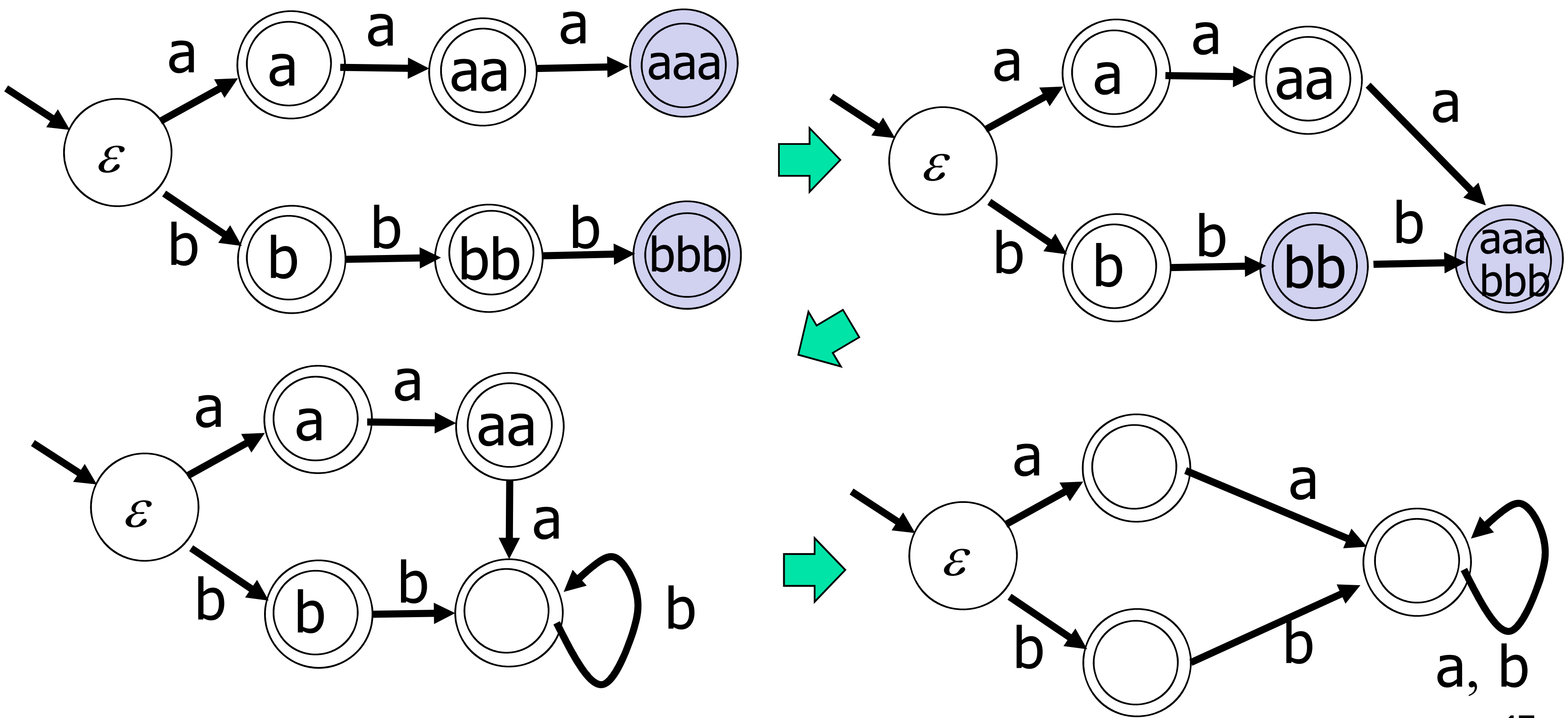
Effect of the Order (1)

- C = {a, b, aa, bb, aaa, bbb}
  D = {ε, ab, ba, aab, aba, abb, baa, bab, bba}
  [bbb, aaa, bb, aa, b, a]



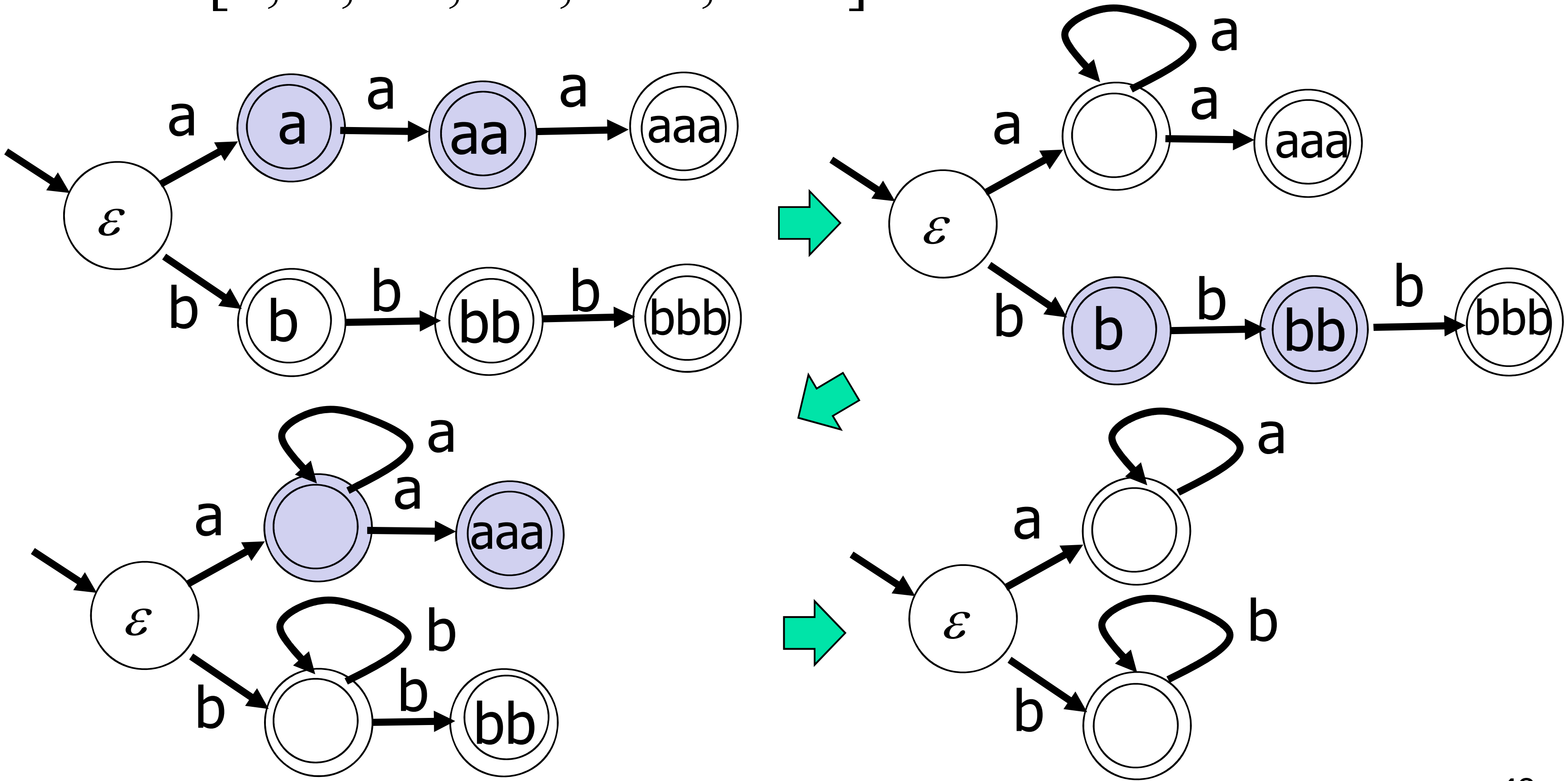
Effect of the Order (2)

- $C = \{$a, b, aa, bb, aaa, bbb$\}$
  $D = \{\varepsilon$, ab, ba$\}$
  [bbb, aaa, bb, aa, b, a]



Effect of the Order (3)

- C = {a, b, aa, bb, aaa, bbb}
  D = {ε, ab, ba}
  [a, b, aa, bb, aaa, bbb]



Effect of the Order (4)

- It is proved that the length-wise lexico-graphic order is better than its inverse.

Finding minimum FA

- Finding a minimum FA consistent with a finite amount of positive and negative examples is NP-hard.
- The automata found by RPNI is not always minimal, but outputs in polynomial time

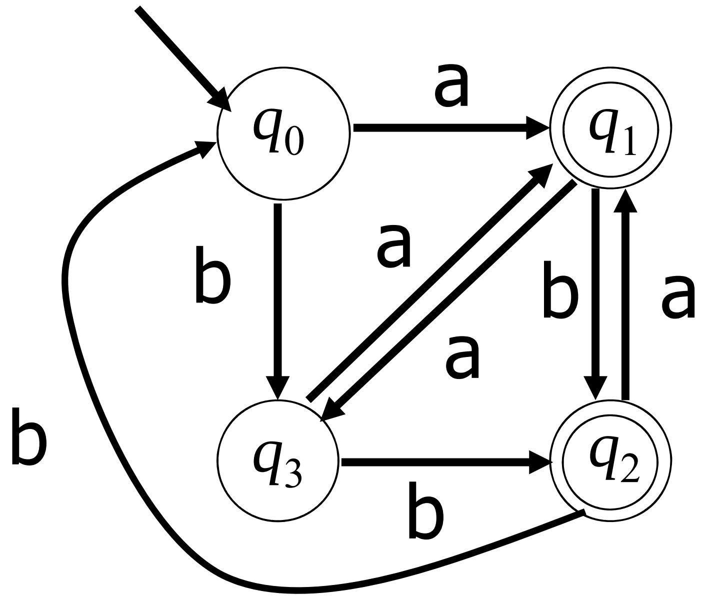
$$\mathrm{card}(C)^2\,\mathrm{card}(D).$$

Data Sets Enough to Output Hidden Automata

Minimal Test Sets

- A set $S \subset \Sigma^*$ is a minimal test set for a FA M if for each state q of M, there exists exactly one string x such that $\delta(q_0, x) = q_i$.

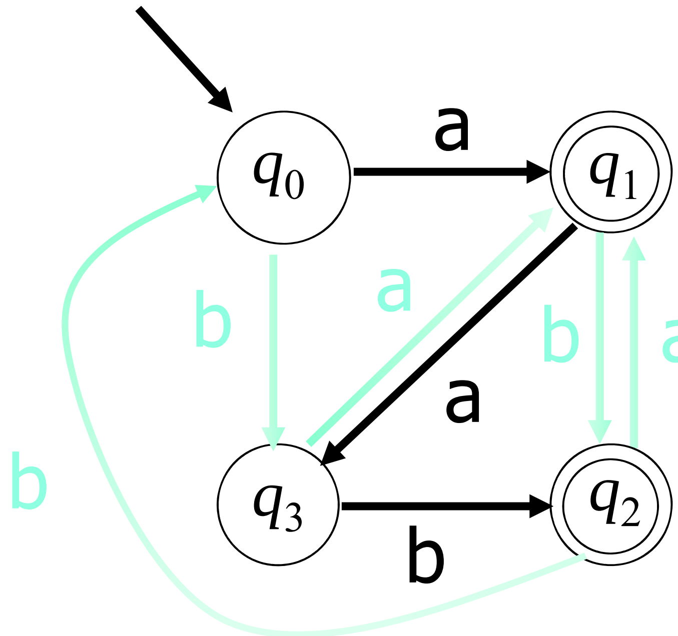
Example Examples of test sets of M are $S_1 = \{\varepsilon$, a, aa, aab$\}$ and $S_2 = \{\varepsilon$, a, ab, b$\}$.



Minimal Test Sets

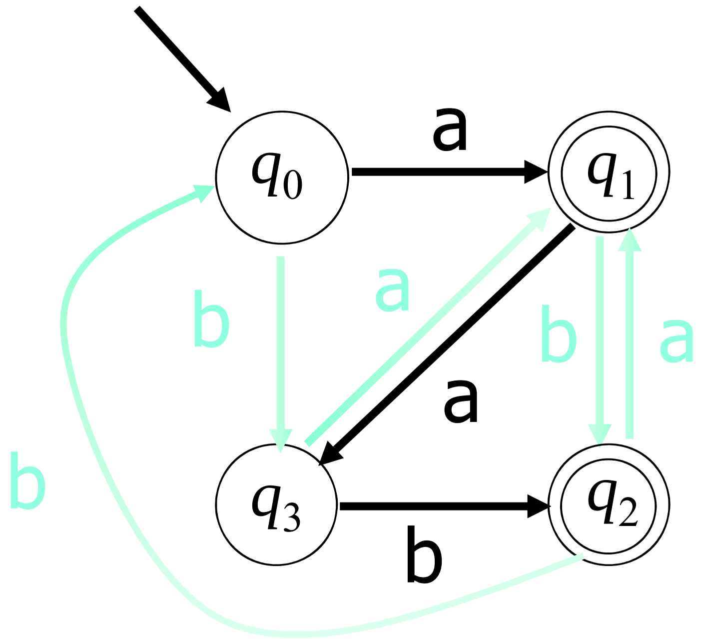
- Intuitively, a test set gives a "skelton" of the finite state automaton.
  - But the set is not sufficient to identify the FA.

Example Examples of test sets of M are $S_1 = \{\varepsilon,$ a, aa, aab$\}$ and $S_2 = \{\varepsilon,$ a, ab, b$\}$.



Prefix closed Test Sets

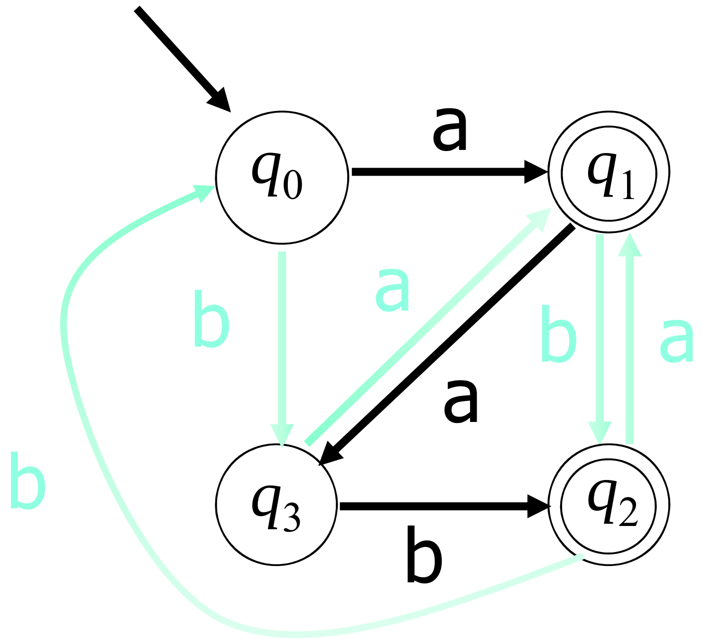
- A set of strings *S* is prefix closed (suffix closed) if and only if every prefix (resp. suffix) of every member of *S* is also a member of *S*.
- Intuitively, a prefix closed minimal test set gives a "skelton" of the finite state automaton.
  - But the set is not sufficient to identify the FA.



Example Both $S_1 = \{\varepsilon$, a, aa, aab$\}$ and $S_2 = \{\varepsilon$, a, ab, b$\}$ are prefix closed.

Prefix closed Test Sets

- A set of strings S is prefix closed (suffix closed) if and only if every prefix (resp. suffix) of every member of S is also a member of S.
- Intuitively, a prefix closed minimal test set gives a "skelton" of the finite state automaton.
  - But the set is not sufficient to identify the FA.

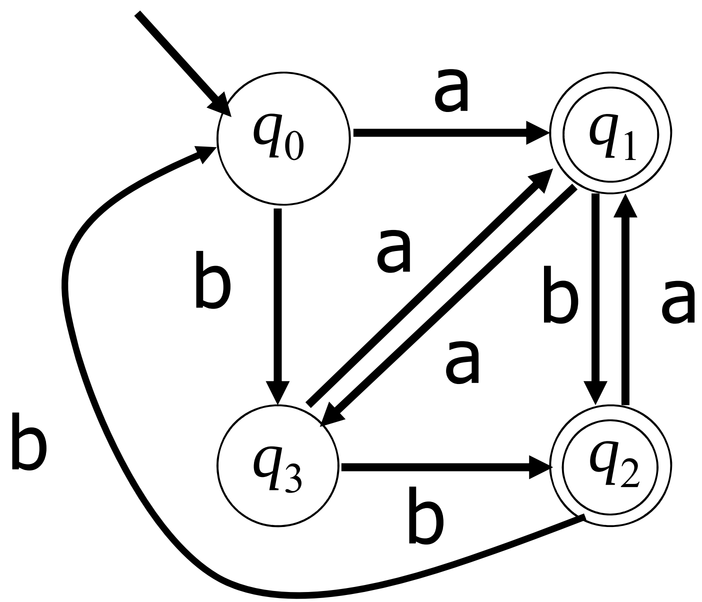


Example Both S_1 = {ε, a, aa, aab} and S_2 = {ε, a, ab, b} are prefix closed.

Fixing an Order

- We fix one ordering for listing elements of a set.

  - Example Following the lexicographic ordering, elements of S_1 = {ε, a, aa, aab} is listed as ε, a, aa, aab

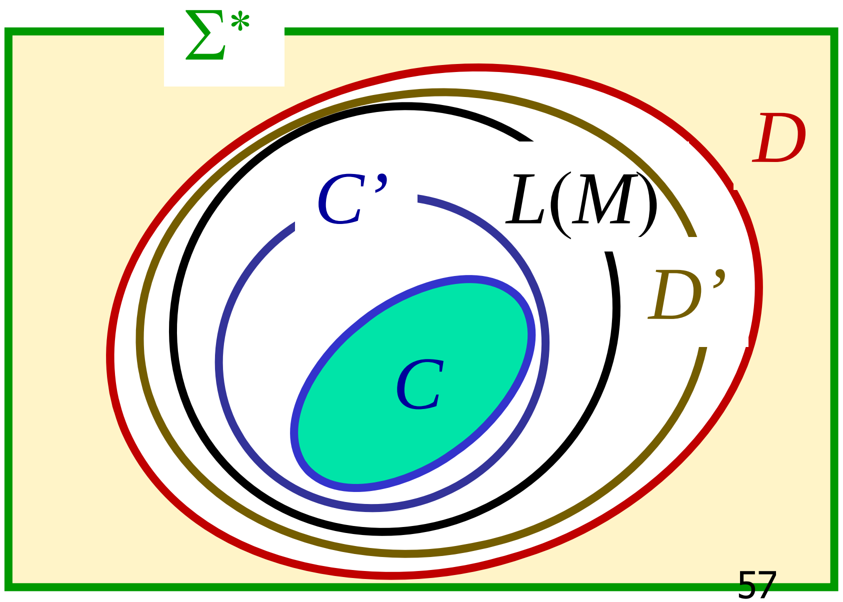


Characteristics Examples

- Assume an algorithm *A* which learns FA.
- Assume that we treat only minimal FA.
- A pair (*C*, *D*) of sets of examples is characteristic for a FA *M* if for any pair (*C*', *D*') of examples such that

$$C \subset C' \subset L(M) \text{ and } D \subset D' \subset \overline{L(M)}$$

the algorithm *A* returns *M*.



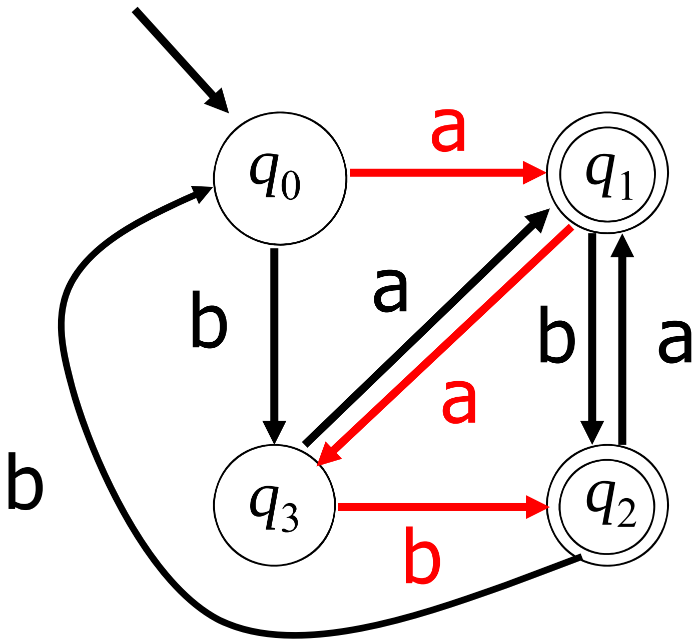
Observation table

- An observation table (S, E, T) :
  S : a prefix closed set $S \subset \Sigma^*$
  E : a suffix closed set $E \subset \Sigma^*$
  $T : (S \cup S\,\Sigma)E \rightarrow \{0, 1\}$
  - $S\,\Sigma = \{\ sa \mid s \in S \text{ and } a \in \Sigma\ \}$
  - The element of the position (s, w) shows whether or not the automaton M accepts sw.

		E	
		ε	b
S	ε	0	0
S	a	1	1
S	aa	0	1
S	aab	1	0
$S\,\Sigma$	b	0	1
$S\,\Sigma$	ab	1	0
$S\,\Sigma$	aaa	1	1
$S\,\Sigma$	aaba	1	1
$S\,\Sigma$	aabb	0	0

Observation table

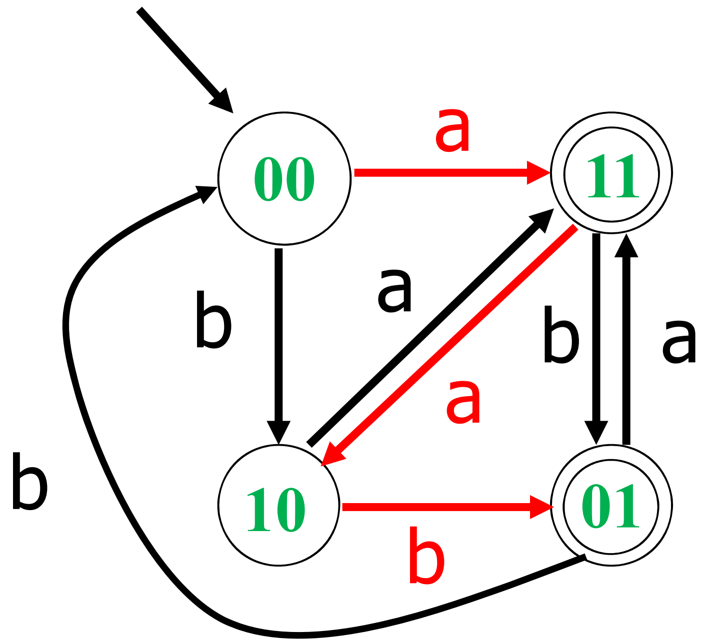
- An observation table (S, E, T) :
  S : a prefix closed set $S \subset \Sigma^*$
  E : a set $E \subset \Sigma^*$
  $T : (S \cup S\,\Sigma)E \rightarrow \{0, 1\}$



	E: ε	E: b
S		
ε	0	0
a	1	1
aa	0	1
aab	1	0
S Σ		
b	0	1
ab	1	0
aaa	1	1
aaba	1	1
aabb	0	0

Observation table

- An observation table (S, E, T) :
  S : a prefix closed set $S \subset \Sigma^*$
  E : a set $E \subset \Sigma^*$
  $T : (S \cup S\,\Sigma)E \rightarrow \{0, 1\}$



		E: ε	E: b
S	ε	0	0
	a	1	1
	aa	0	1
	aab	1	0
$S\,\Sigma$	b	0	1
	ab	1	0
	aaa	1	1
	aaba	1	1
	aabb	0	0

How to construct the table

Input : a minimal FA A

Output : The characteristic set of polynomial size

S : = the minimal test set of A, $E := \{ \varepsilon \}$, $S' := S\Sigma - S$,

Generate (S, E, T);

while there exists $w, v \in S$ s.t. row(w) = row(v) but

$T(wc, e) \neq T(vc, e)$ for some $c \in \Sigma$ and $e \in E$

$E : = E \; \{ae\}$;

Generate (S, E, T);

end while

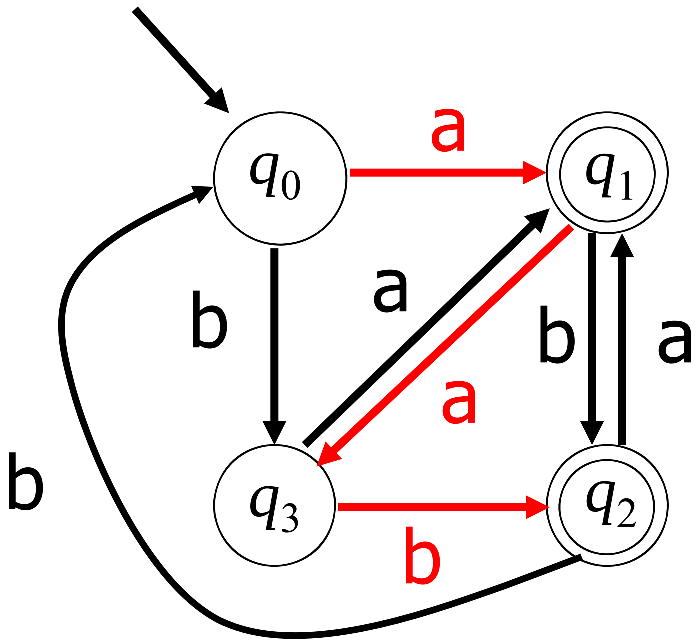
$C = \{ \, we \mid \; w \in S \cup S\Sigma, e \in E, \text{ and } T(wc, e) = 1 \}$

$D = \{ \, we \mid \; w \in S \cup S\Sigma, e \in E, \text{ and } T(wc, e) = 0 \}$

return (C, D);

Example

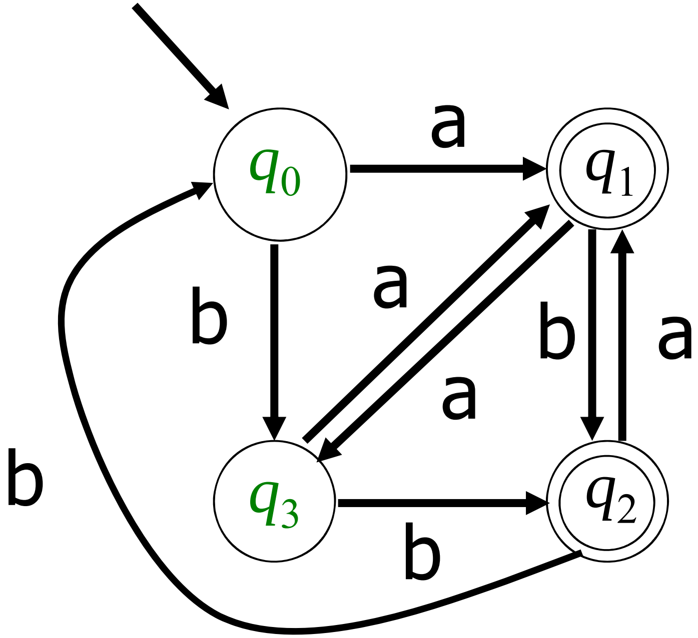
- $S = \{\varepsilon$, a, aa, aab$\}$
- $S\Sigma = \{$a, aa, aaa, aaba
  b, ab, aab, aabb$\}$
- $E = \{\varepsilon\}$.



		E: ε
S	ε	0
S	a	1
S	aa	0
S	aab	1
$S\Sigma$	b	0
$S\Sigma$	ab	1
$S\Sigma$	aaa	1
$S\Sigma$	aaba	1
$S\Sigma$	aabb	0

Example

Because $T(\varepsilon, \varepsilon) = T(\text{aa}, \varepsilon)$, check whether or not $T(\text{a}, \varepsilon) = T(\text{aaa}, \varepsilon)$, and
whether or not $T(\text{b}, \varepsilon) = T(\text{aab}, \varepsilon)$.

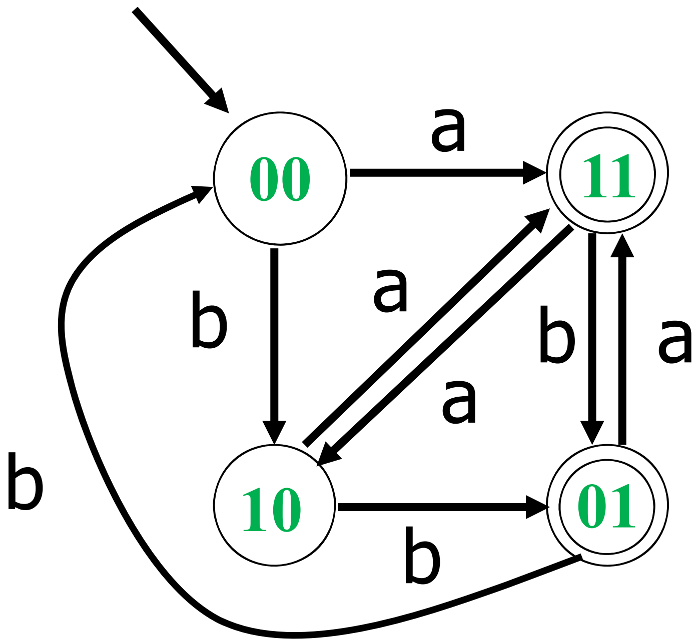


E

		ε
S	ε	0
	a	1
	aa	0
	aab	1
$S\,\Sigma$	b	0
	ab	1
	aaa	1
	aaba	1
	aabb	0

Example

- $E := E \cup \{b\}$
- Fill all of the new elements of the extended table.

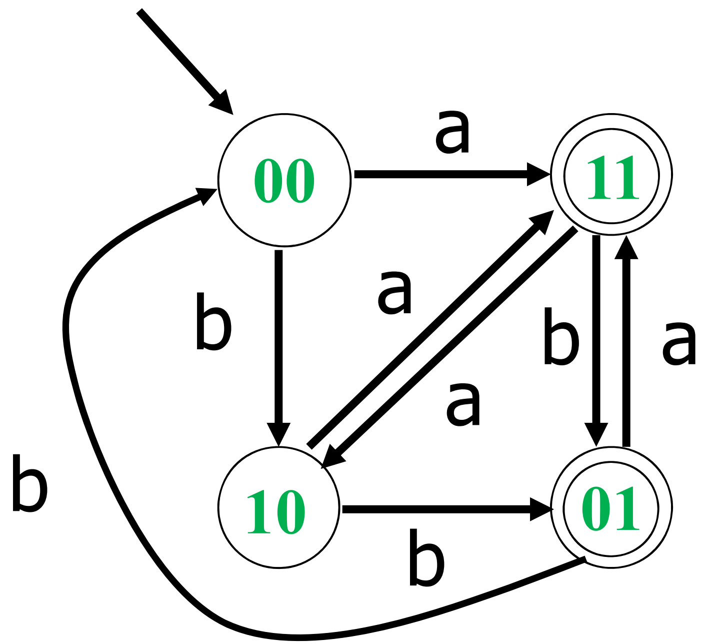


	ε	b
ε	0	0
a	1	1
aa	0	1
aab	1	0
b	0	1
ab	1	0
aaa	1	1
aaba	1	1
aabb	0	0

Example

- There is no *w* and *v* in the *S* part s.t. row(*w*) = row(*v*), end the loop.
- C = {a, ab, bb, aaa, aab, aaab, aaba, aabab}

  D = {ε, b, aa, abb, aabb, aabbb}



		ε	b
S	ε	0	0
	a	1	1
	aa	0	1
	aab	1	0
$S\Sigma$	b	0	1
	ab	1	0
	aaa	1	1
	aaba	1	1
	aabb	0	0

Consistent Table

- An observation table (S, E, T) is consistent if and only if for every pair $w, v \in S$ such that $\mathrm{row}(w) = \mathrm{row}(v)$, $\mathrm{row}(wc) = \mathrm{row}(vc)$ for any $c \in \Sigma$.
  - Intuitively, in a consistent table, every row in the S part can be regarded as one state of an automaton.

Proposition A consistent table T represents an automaton M such that, for $w \in S \cup S\,\Sigma$ and $e \in E$, M accepts we if and only if $T(w, e) = 1$.

Characteristic Examples

Theorem Suppose T be the table obtained above method from M. Then the pair (C, D) where

$$C = \{we \mid w \in S \cup S\,\Sigma \text{ and } e \in E \text{ and } T(w, e) = 1\}$$

$$D = \{we \mid w \in S \cup S\,\Sigma \text{ and } e \in E \text{ and } T(w, e) = 0\}$$

is characteristic w.r.t. the generate-and-test algorithm and M.

The Myhill-Nerode Theorem

Theorem The following three statements are equivalent:
(1) The language L is accepted by some finite automaton.
(2) L is the union of some equivalence classes of a right invariant equivalence relation of finite index.
(3) Let equivalence relation R_L be defined by: $x\, R_L\, y$ if and only if for all $z \in \Sigma^*$ xz is in L iff yz is in L. Then R_L is finite index.

- An equivalence relation R is right invariant iff $x\, R\, y$ implies $xz\, R\, yz$ for all $z \in \Sigma^*$.
- The index of equivalence relation R is the number of equivalence classes.