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

Machine Leaning from String Data

Alphabets and Stings

- Σ : a finite set of symbols and called an alphabet
- Σ^* : the set of all finite strings (sequences) consisting of the symbols in Σ.
  - An empty string is denoted by ε.
  - $\Sigma^+ = \Sigma^* - \{\varepsilon\}$
  - The size of a string w, denoted by $|w|$, is the total number of symbols occurring in w.

Examples

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

  $\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$

  $|aaa| = |aab| = 3$, $|a| = |b| = 1$, $|\varepsilon| = 0$
- $\Sigma = \{A, T, C, G\}$

  $\Sigma^* = \{\varepsilon, A, T, C, G, AA, \ldots, AG, TA, \ldots, AAA, \ldots\}$

Question

- Assume that we have provided

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

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

  such that $C \cap D = \emptyset$.

- Develop a computer program to find a rule which accepts all positive examples and rejects all negative examples.

Examples

Example 1

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

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

- It could hold that *every string in C_1 starts with* a *and end with* b.

Example 2

$C_2 = \{$ba, bababa, babababa, bababababa$\}$

$D_2 = \{$a, b, bbbb, abb, baaaaba, babbb$\}$

- It might hold that *every string in C_2 is made of some repetition of* ba.

Examples

Example 3

C_3 ={aaabbb, ab, aaaabbbb, aaaaabbbbb, aabb}

D_3 ={a, b, bbbb, abb, baaaaba, babbb}

- *Every string in C_3 consists of two strings: The first string consists only of* a*'s, and the second consists of the same number of* b*'s.*

Example 4

C_4 = {aa, abaaba, aaaaaaa, baaab, abab}

D_4 = {a, b, bbbb, abb, bbbbbbba, babbb}

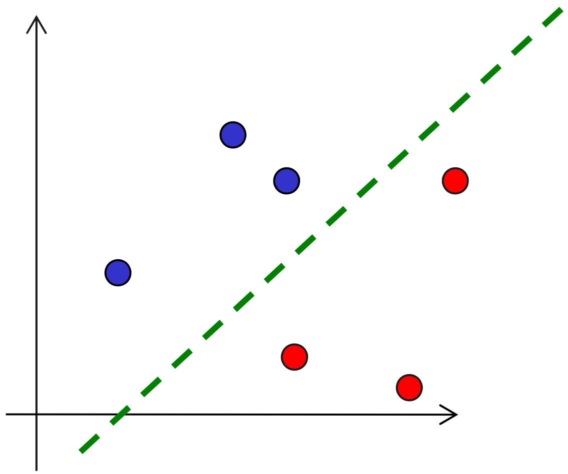
- *In every string in C_4 has more than two* a*'s.*

The First Problem

- What is the grammar and vocabulary with which we represent the rule to distinguish C and D?
  - In the linear classification case, the rule to be found is represented in the form of $(\boldsymbol{w}, \boldsymbol{x}) + c$ s.t.

$$\boldsymbol{x} \in C \Rightarrow (\boldsymbol{w}, \boldsymbol{x}) + c \geq 0$$

$$\boldsymbol{x} \in D \Rightarrow (\boldsymbol{w}, \boldsymbol{x}) + c \leq 0$$



- The region including C is represented with an inequation

$$(\boldsymbol{w}, \boldsymbol{x}) + c \geq 0$$

Solutions to the Problem

- We adopt some representation method with which we represent a subset of Σ^* which includes C.
  - Since the rule found by some learning mechanism is expected to be "general", the set should be sufficiently large.

    Rules should not overfit the examples.
  - A rule which represents a rule is sometimes called a predicate.

Examples

Example 1

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

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

- The rule which is output by a learning machine would represent a set

L_1 = {ab, aab, abb, aaab, aabb, abab, abbb,
aaaab, aaabb, ..., abaab, ..., abbbb,
..., aaaabbbb, ...}

Examples

Example 3

C_2 ={aaabbb, ab, aaaabbbb, aaaaabbbbb, aabb}

D_2 ={a, b, bbbb, abb, baaaaba, babbb}

- You may imagine that the rule which is output by a learning machine would represent a set

L_2 = {ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, aaaaaabbbbbb,...}

Formal Languages

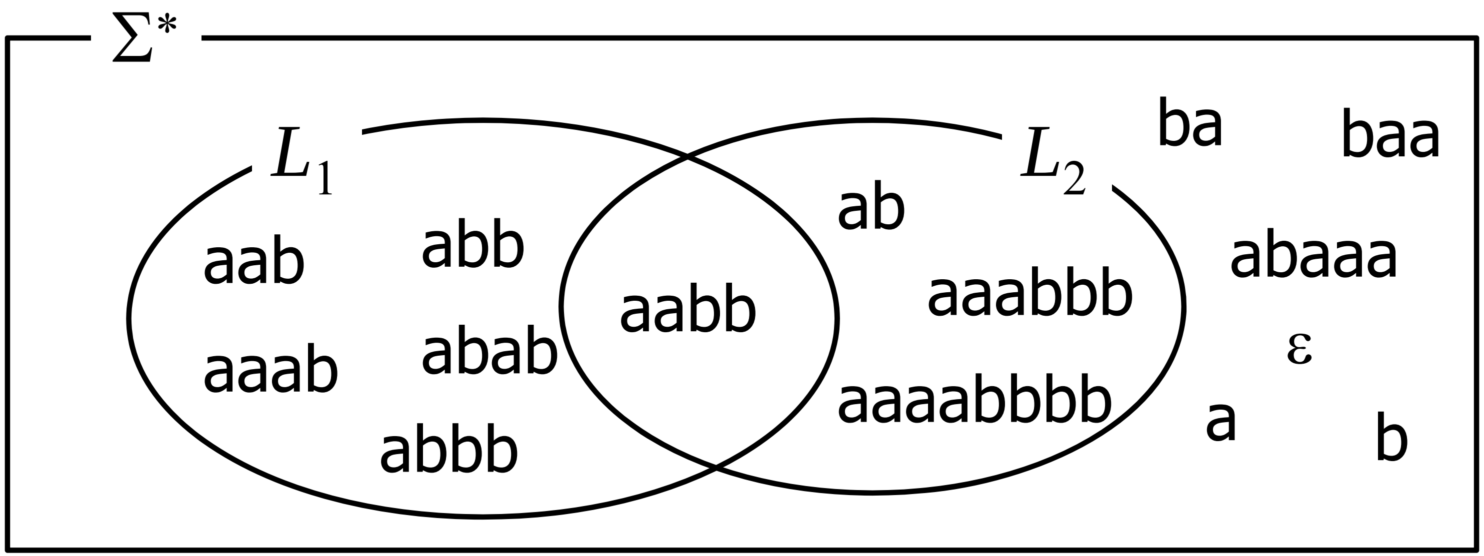
- Every subset of Σ^* is called a formal language.

Example

$\Sigma = \{a, b\}$, $\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$

$L_1 = \{aab, abb, aaab, aabb, abab, abbb, \ldots\}$

$L_2 = \{ab, aabb, aaabbb, aaaabbbb, \ldots\}$



Learning by Enumerating FA

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$\}$

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 $\boldsymbol{x} = \langle w, s \rangle$ of a string with a sign such that $s = +$ if $w \in C$ and $s = -$ if $w \in D$,

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

$\mathrm{P}(M)$: 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,

  $\delta : S \times \Sigma \to S$ is a transition function

  represented as a transition table,

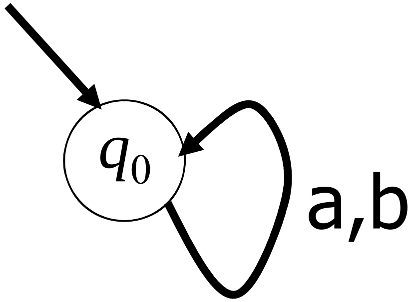
  $q_0 \in S$ is an initial state,

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

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

Finite Automata of One State

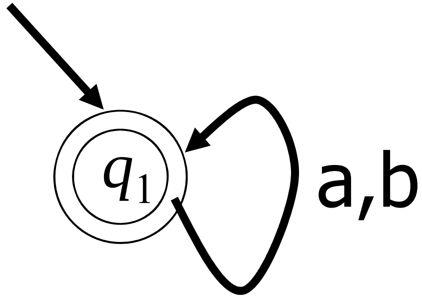
M_0



	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

    $M_1, M_2, \ldots$

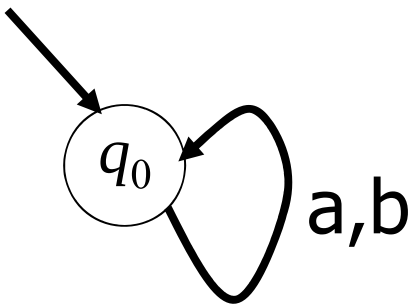
    any automaton M appears as $M_i = M$.

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

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

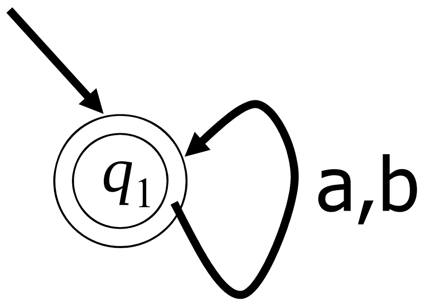
Enumeration of Automata(1)

M_0



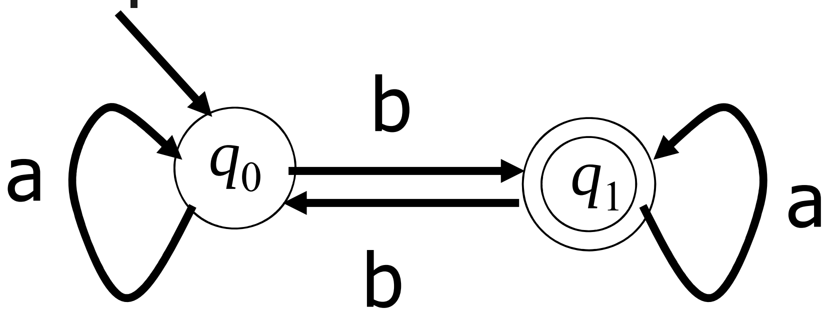
	F	a	b
q_0		q_0	q_0

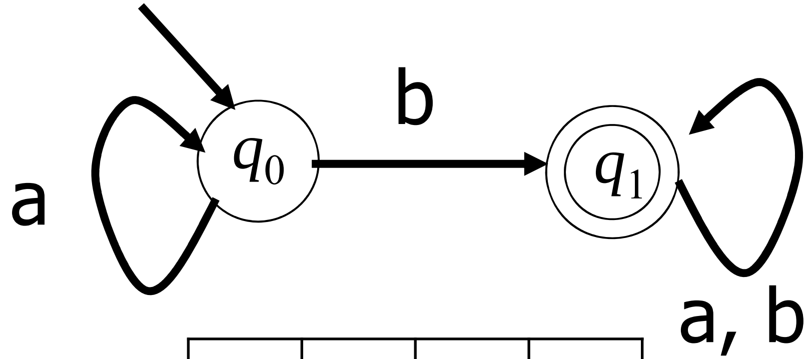
M_1



	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		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_0

	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 ($C \cap D = \varnothing$), there exists a finite state automaton M such that $L(M) = C$ and $L(M) \cap D = \varnothing$, 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

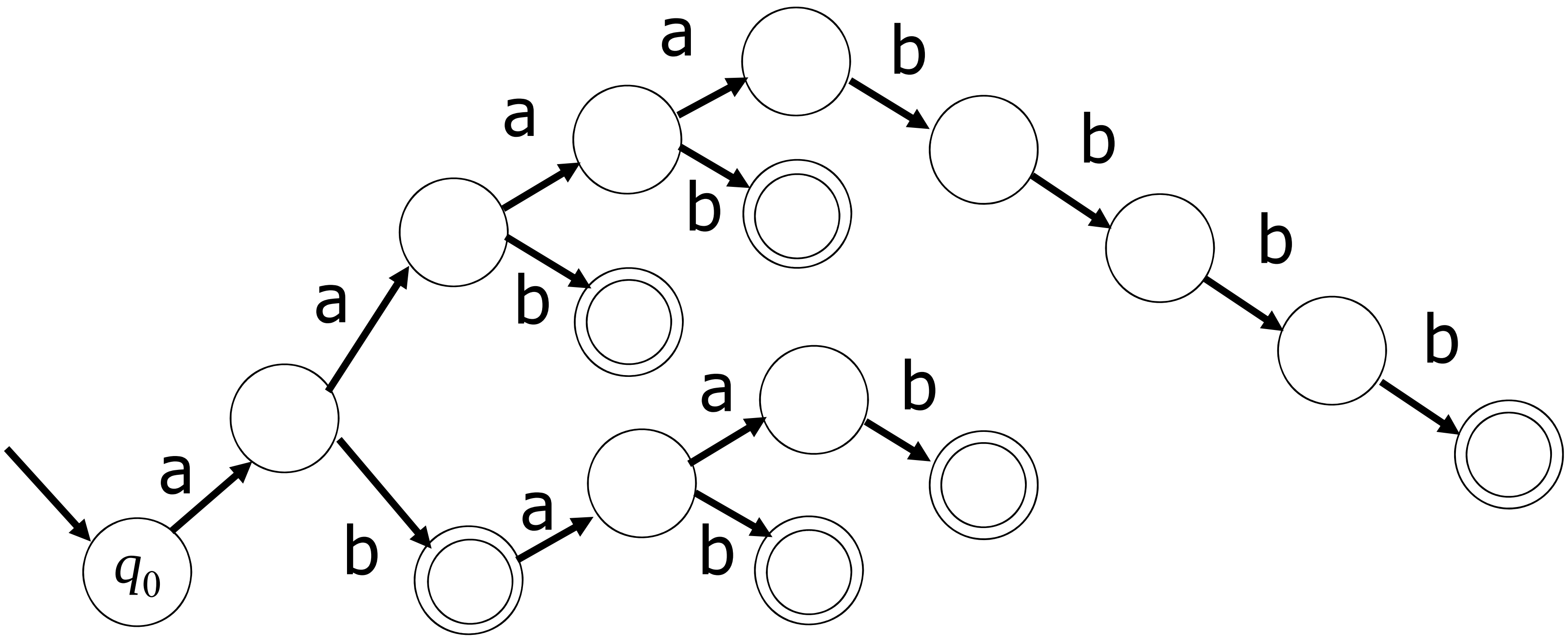
$$L(M) \subset C \text{ and } L(M) \cap D = \varnothing.$$

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

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

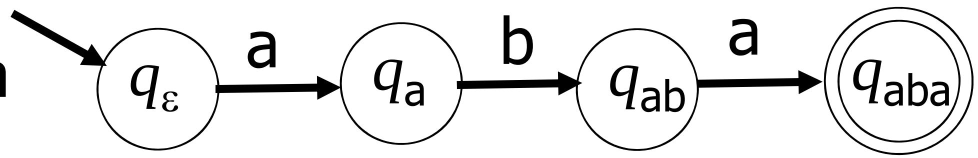
where

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

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

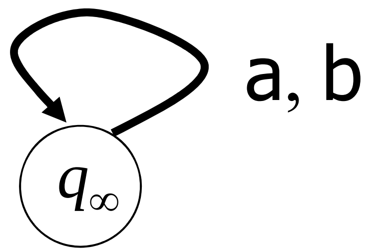
$$Q_S = \{ q_s \mid s \in S \}$$

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".

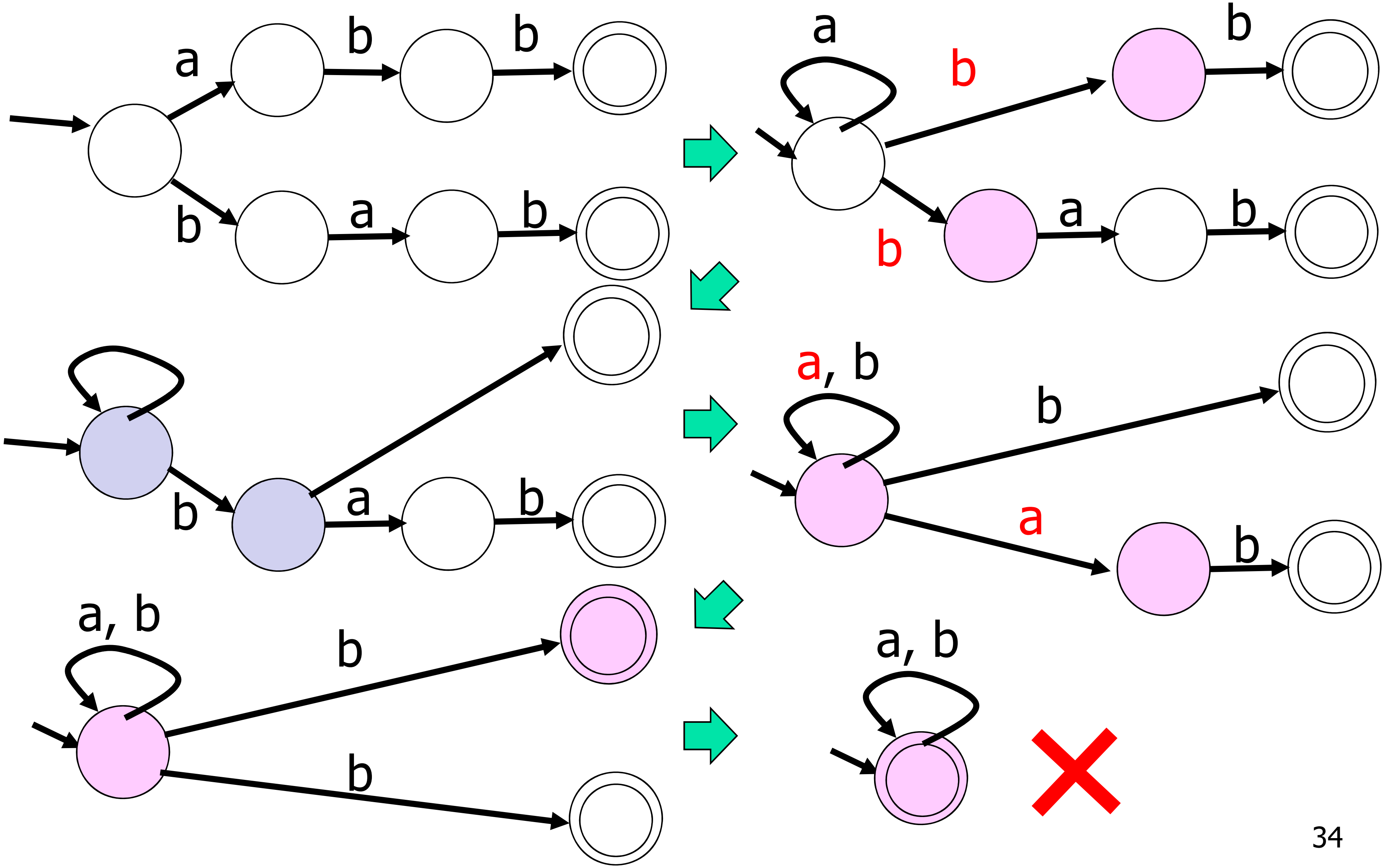
Generalization by Merging States

Generalization by Merging States

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

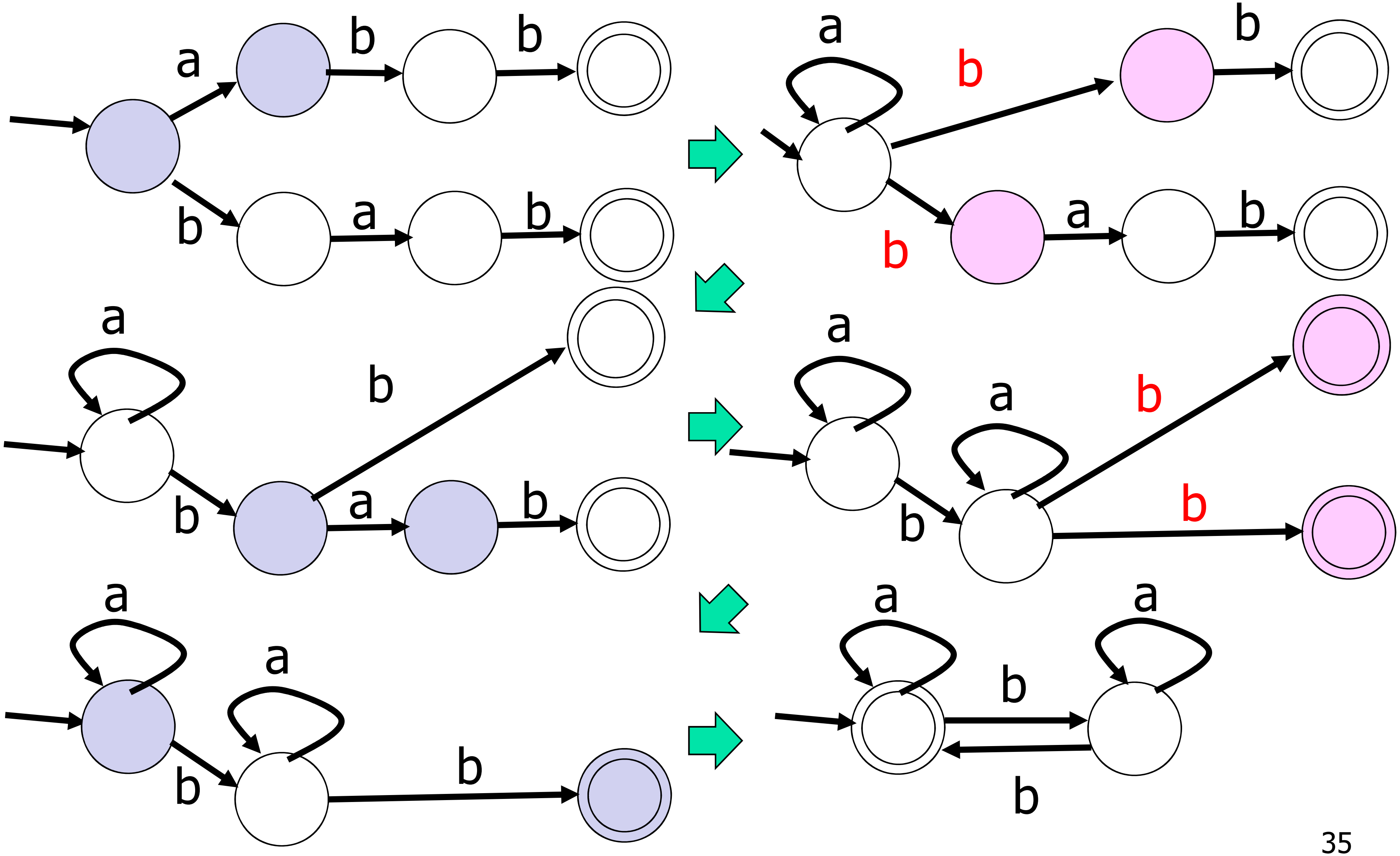
Example: Merging States

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



Example: Merging States(cont.)

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



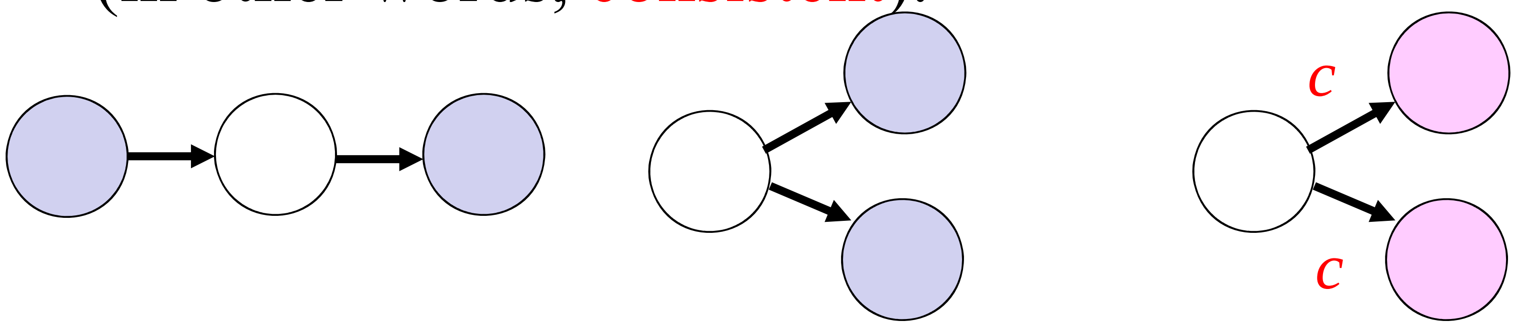
Two Types of Merge

- 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

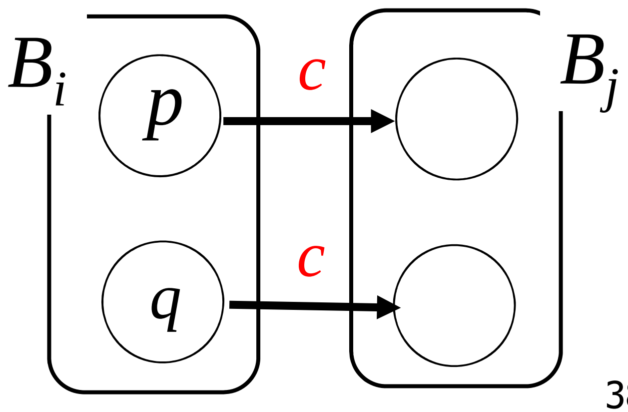
$\Leftrightarrow$

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 $P(C)$

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

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

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

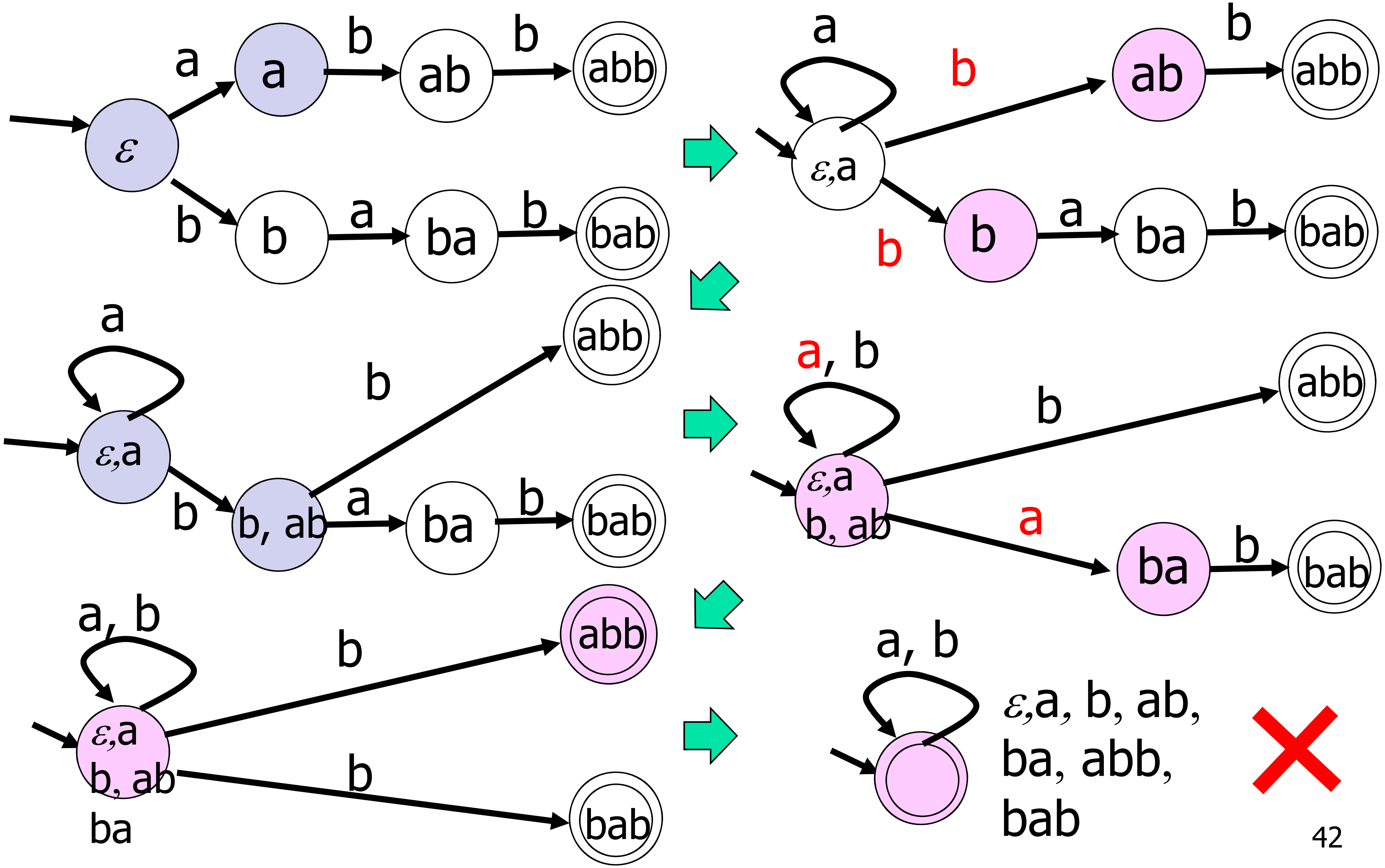
- We use the length-wise lexico-graphic order:

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

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

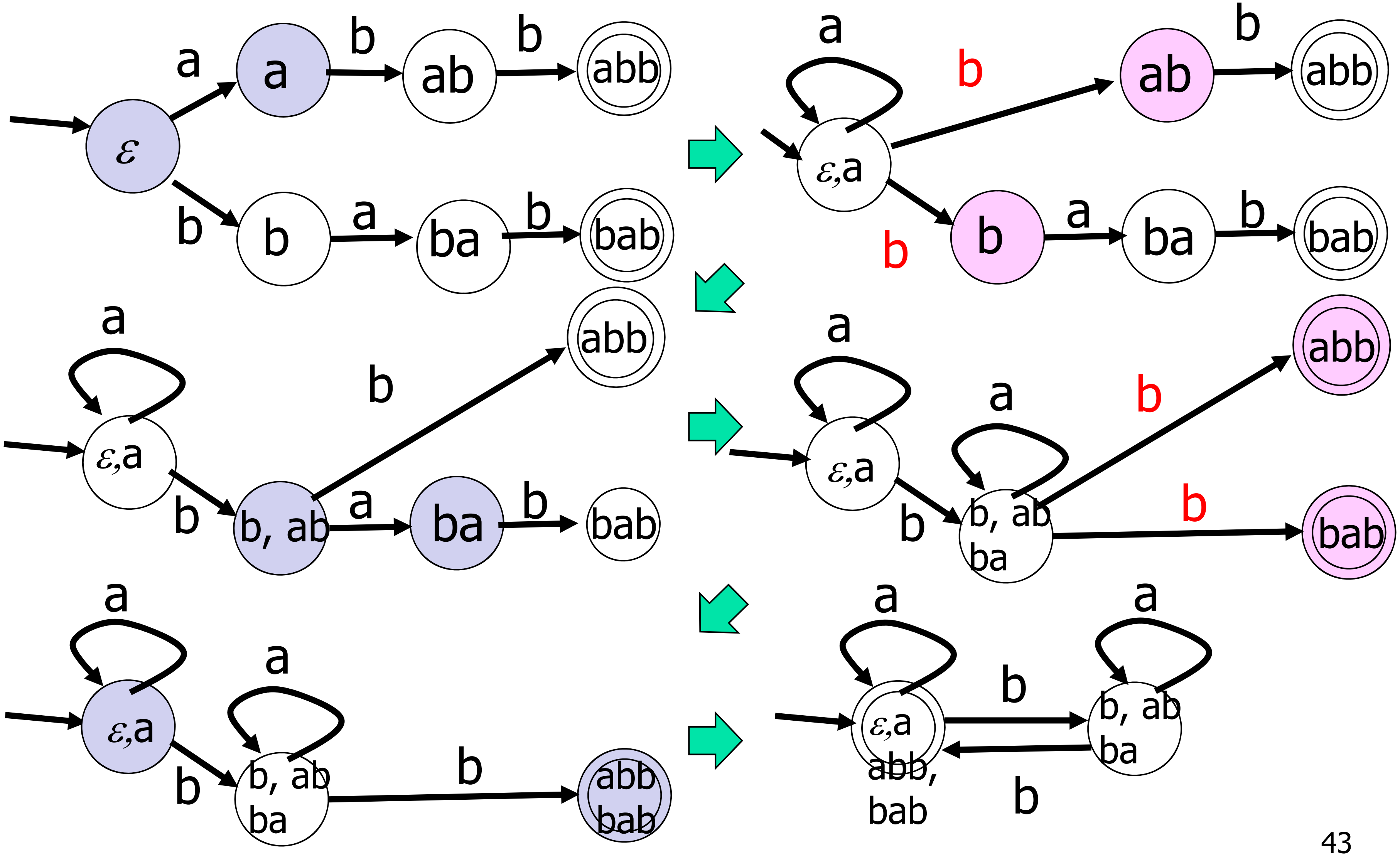
Example: Merging States

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



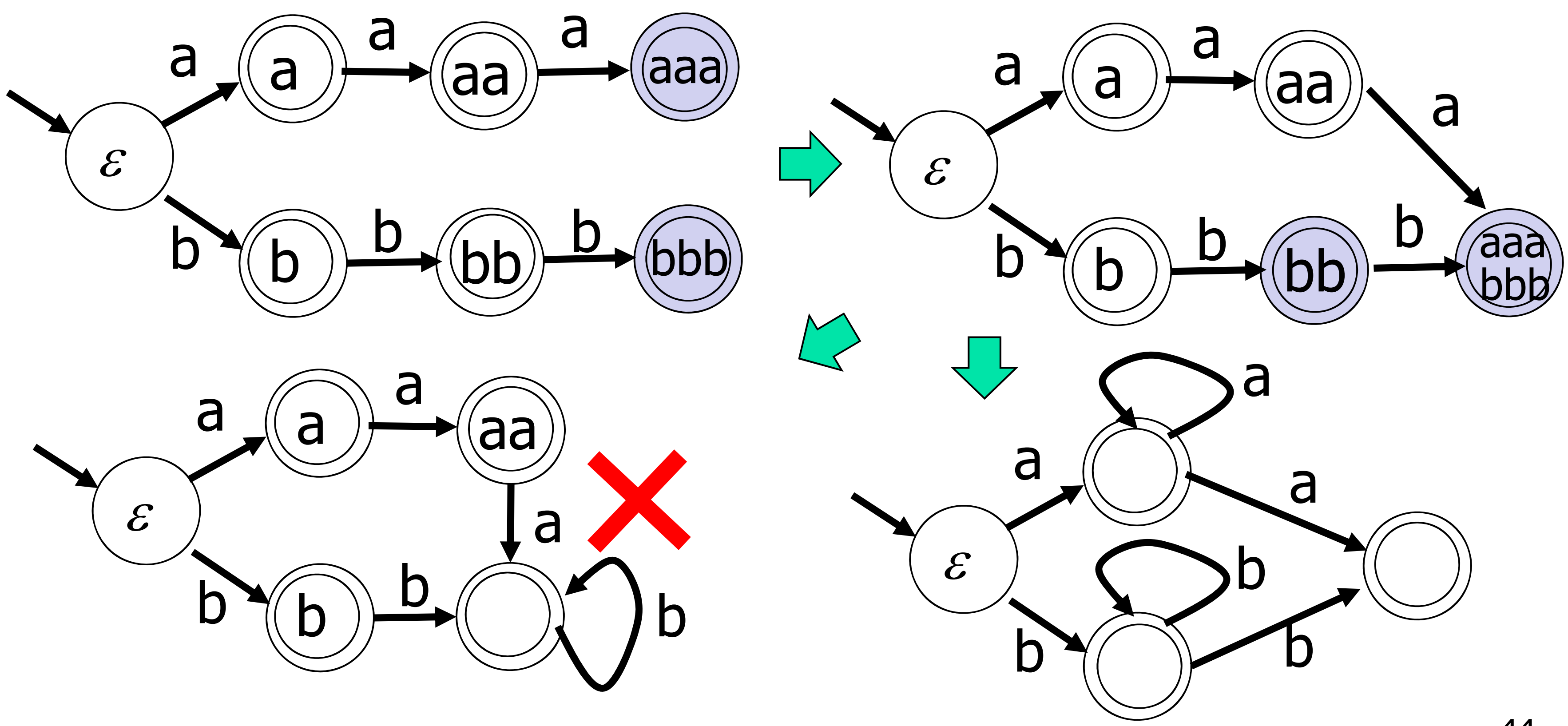
Example: Merging States(cont.)

C = {abb, bab}, D = {ab}



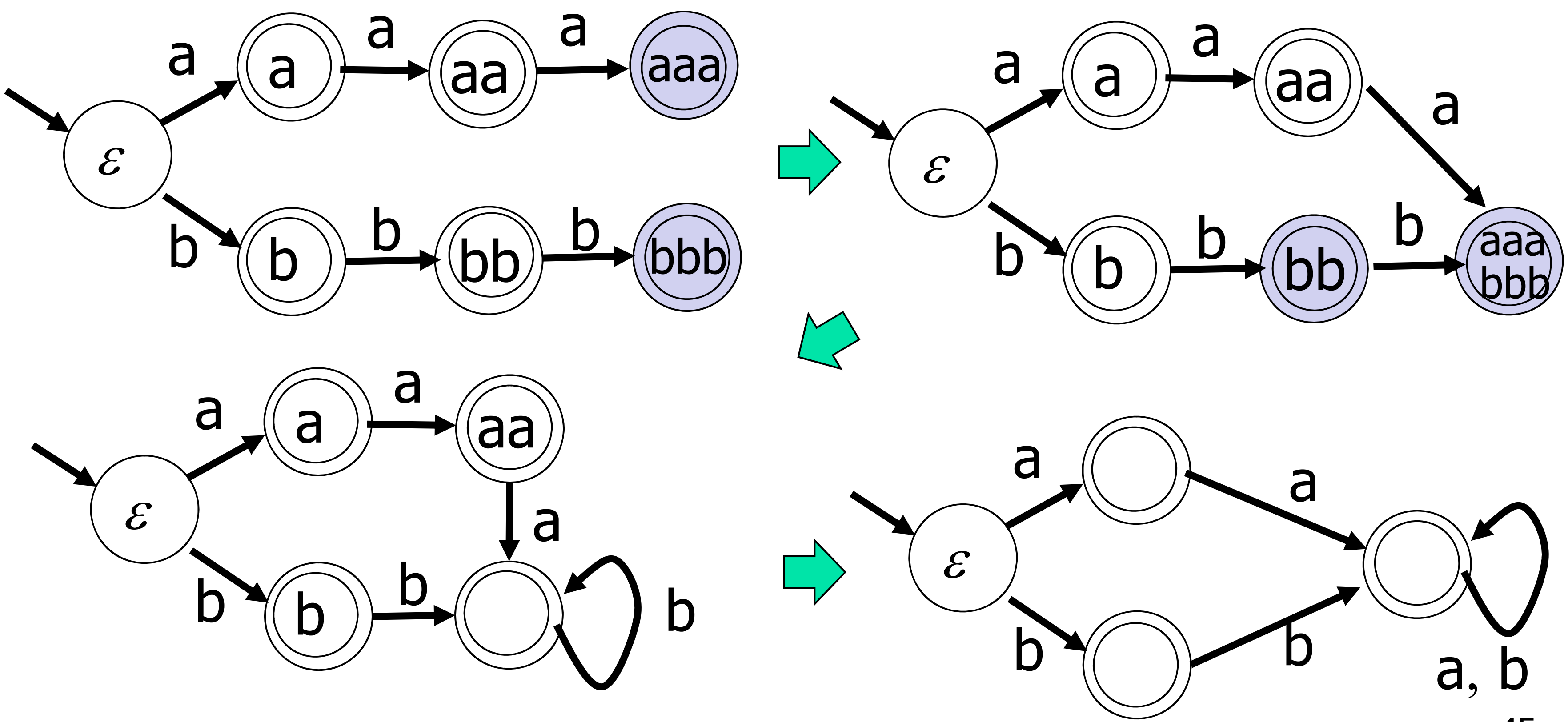
Effect of the Order (1)

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



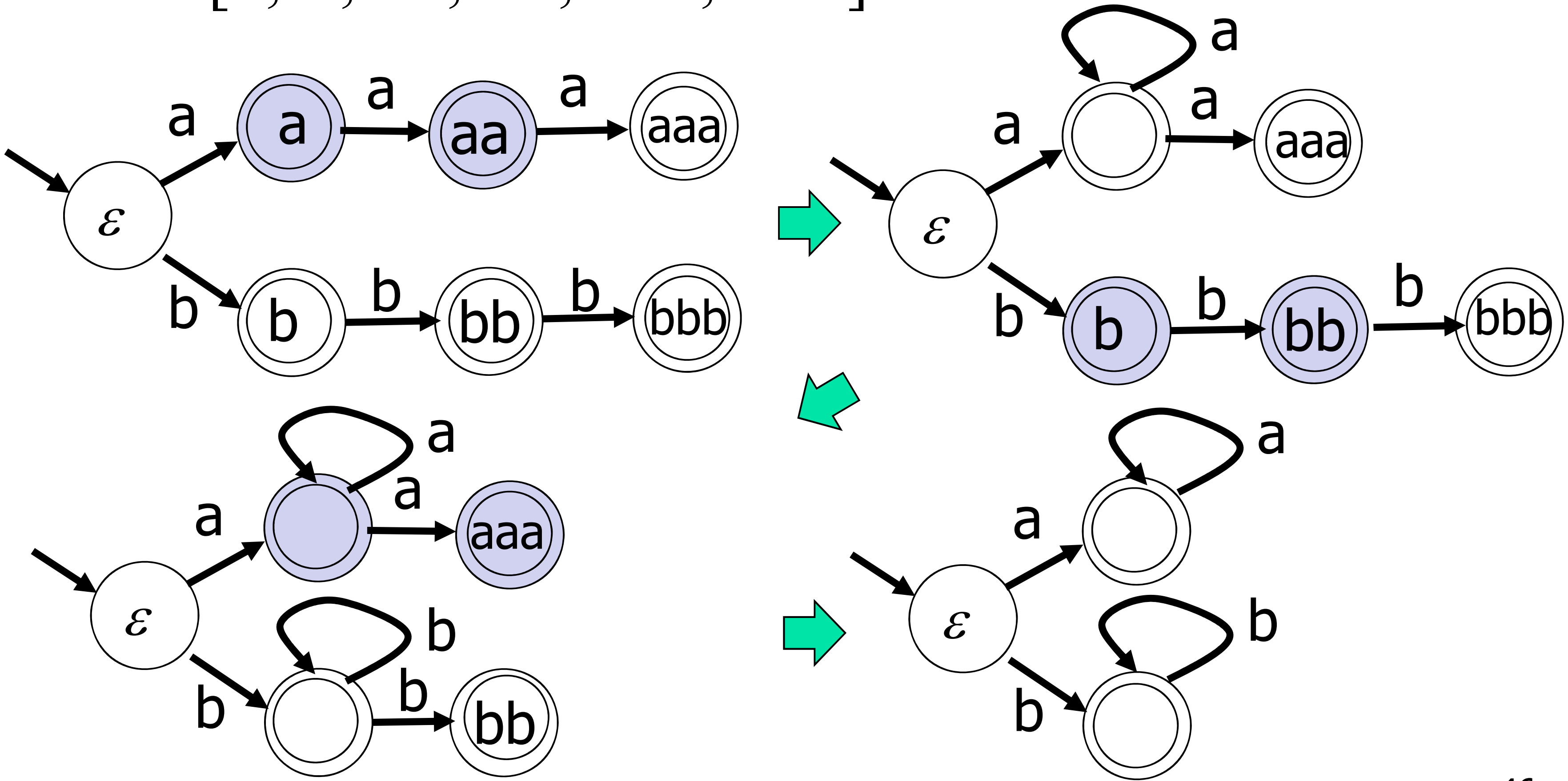
Effect of the Order (2)

- $C = \{$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

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