Computational Learning Theory

Correctness of Algorithms for 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

Note

- There might be several automata consistent with given *C* and *D*.
- In such automata 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".

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 $x_n \in L(M_{k'}))$

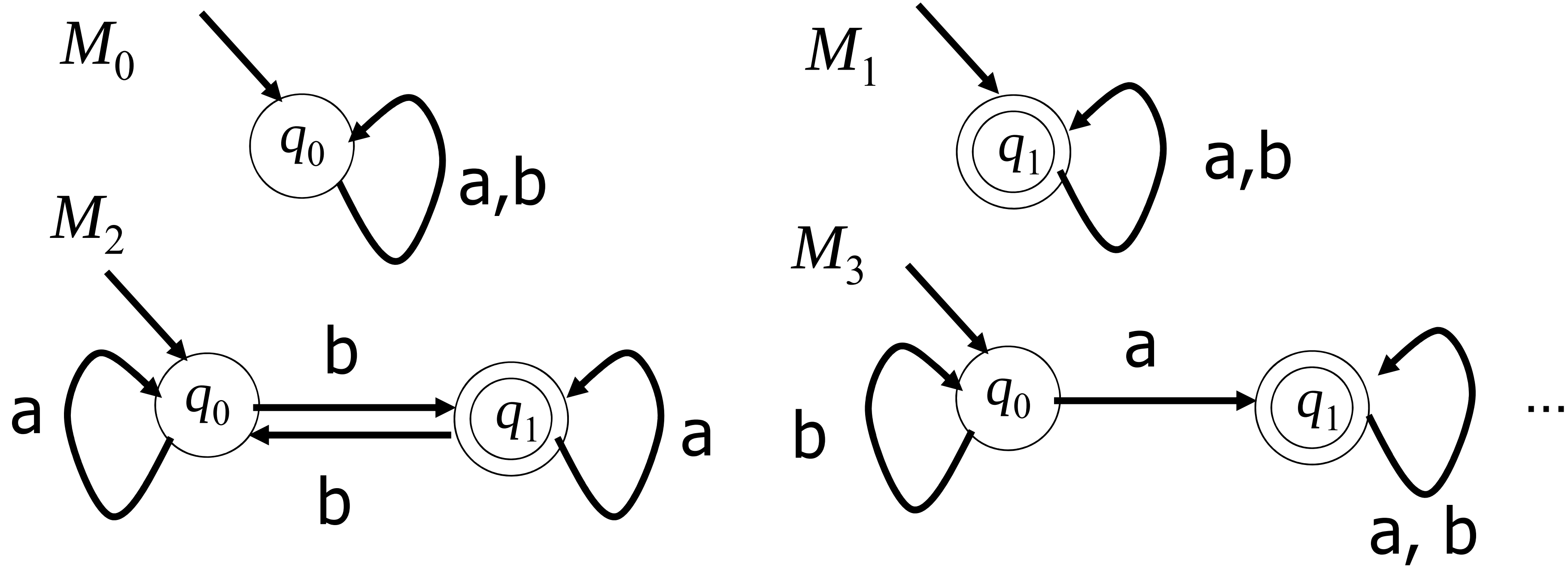
      replace k with $k + 1$

  if $k' = k$

    terminate and output M_k

Example

- Enumeration of FAs



Input:

σ : <ab,+>, <aab,+>, < bbb,–>, <aaab,+>, <abba,–>,…

Output:

M_1 M_1 M_3 M_3 …

Example(cont.)

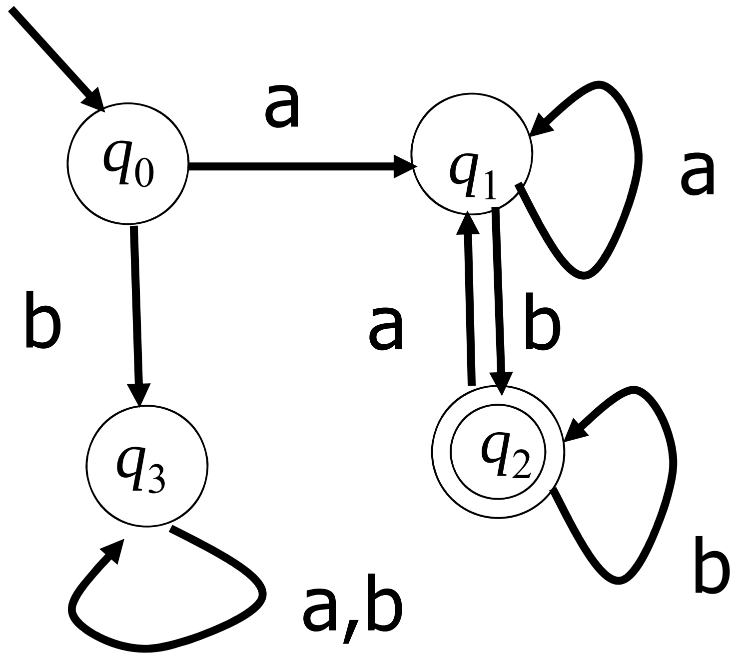
Input:

σ : …, < bbb,–>, <aaab,+>, <abba,–>,… … … …

Output:

… M_3 M_3 … … M' M' M'

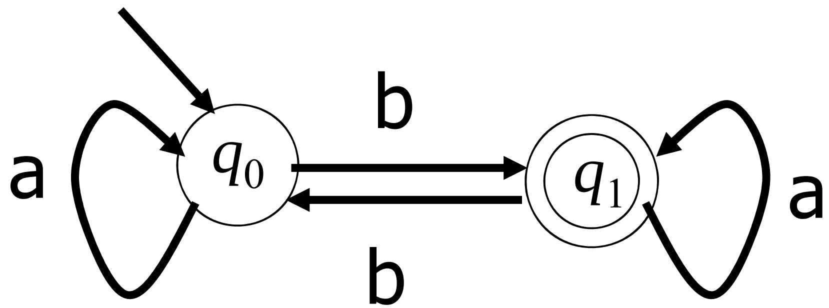
M

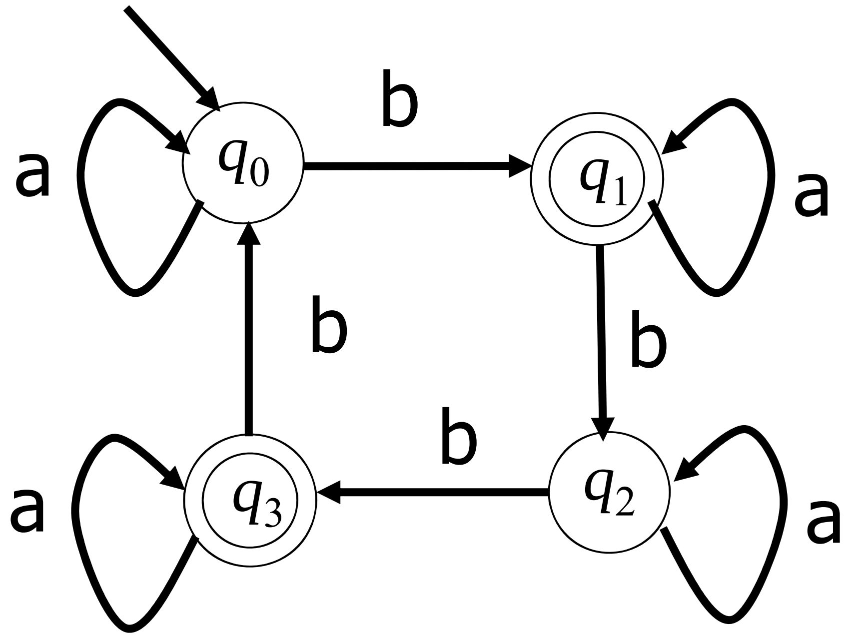


$L(M') = L(M)$

Note

- For a finite automaton M, there might be exists another automaton such that $L(M) = L(M')$.





Evaluation of Learning Algorithm

Support Vector Machine

- **Input:** a set of numerical data

  $$\{(\boldsymbol{x}_1, c_1), (\boldsymbol{x}_2, c_2), \ldots, (\boldsymbol{x}_m, c_m)\} \quad \boldsymbol{x}_i \in \boldsymbol{R}^n$$

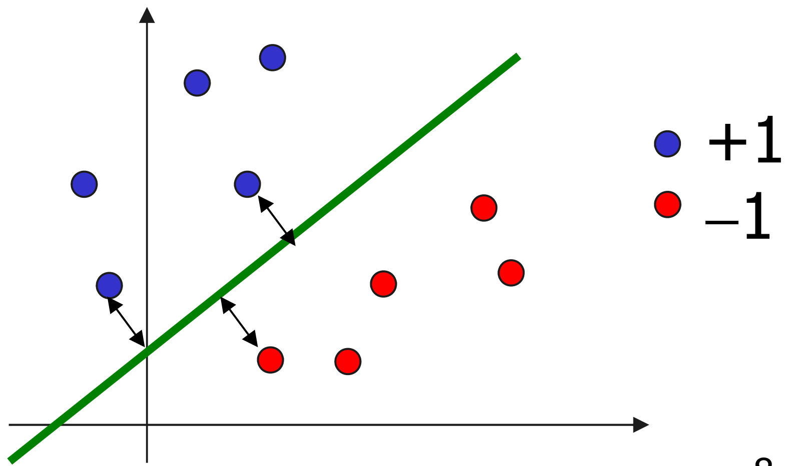
  where each $c_i \in \{+1, -1\}$ is a class signal for $\boldsymbol{x}_i$

  Output: find a liner function (hyper-plane)

  $$f(\boldsymbol{x}) = \Sigma\, w_i\, \boldsymbol{x}_i \cdot \boldsymbol{x} + b$$

  which $sign\,(f(\boldsymbol{x}_i)) = y_i$ for all i and

  maximize the margin $\min_{1 \le i \le m} \mathrm{d}(f, \boldsymbol{x}_i)$



General Learning

- Recently a machine learning method is recognized as one to find

  $$\text{argmin}_{f \in H} \left(\Sigma_{\boldsymbol{x} \in D} \text{Loss}(f, \boldsymbol{x}) + \lambda \, \text{P}(f)\right)$$

  where

  Loss(f, $\boldsymbol{x}$) is a loss function and

  P(f) = is a penalty function.

  - This definition is declarative.
  - This course we introduce some of the instances of Loss(f, $\boldsymbol{x}$) and P(f).

Abstract Classification

- A half-plane P which contains C (yes) and excludes D (no) is to be learned
- The half-plane P is represented as a pair $(\boldsymbol{w}, c)$ which means the linear inequation $(\boldsymbol{w}, \boldsymbol{x}) + c > 0$.
  - Let $C(p)=\{\boldsymbol{x} \in \boldsymbol{R}^n \mid p(\boldsymbol{x})\}$ for a predicate p.

    Then the search space (version space) is

    $\mathrm{C} = \{C(\lambda\, \boldsymbol{x}.((\boldsymbol{w}, \boldsymbol{x}) + c > 0)) \mid \boldsymbol{w} \in \boldsymbol{R}^n, c \in \boldsymbol{R}^n\}.$

    The set of parameter s are from

    $\mathrm{H} = \{(\boldsymbol{w}, c) \mid \boldsymbol{w} \in \boldsymbol{R}^n, c \in \boldsymbol{R}^n\}.$
- The training examples are provided as the sets C and D.
- A learning algorithm is provided.

Typical evaluation method

- A learning algorithm A is evaluated with test data as follows.

Step1. Let C_* are set of all positive data and D_* be are all negatives.

Step 2. Select subsets $C_{\text{training}} \subset C_*$ and $D_{\text{training}} \subset D_*$ for training.

Step 3. Apply A to the pair C_{training} and D_{training} and obtain a rule f.

Step 4. Select subsets C_{test} and D_{test} make a confusion matrix.

Step 5. Calculate some measures from the confusion matrix.

Confusion Matrix

- Every data is represented as a pair $\mathbf{x}$ = <w, s>

  $s = +$ if $w \in C$ and $s = -$ if $w \in D$

	C_{test}	D_{test}
$\{w\in C_{\text{test}}\cup D_{\text{test}} \mid f(w) = 1\}$ positive	true positive	false positive
$\{w\in C_{\text{test}}\cup D_{\text{test}} \mid f(w) = 1\}$ negative	false negative	true negative

Measures

- Accuracy
$$\frac{|TP| + |TN|}{|TP| + |FP| + |TN| + |FN|}$$
- Error rates 1- Accuracy
- Precision (positive predictive rate)
$$\frac{|TP|}{|TP| + |FP|}$$
- Recall (coverage, true-positive rate, sensitivity)
$$\frac{|TP|}{|TP| + |FN|}$$

Comparison with an Unknown Function

- Assuming an unknown discriminant function f_* such that

$$C_* = \{ \boldsymbol{x} = <w, 1> \ | \ f_*(w) = 1 \}$$

$$D_* = \{ \boldsymbol{x} = <w, 1> \ | \ f_*(w) = 0 \}$$

we evaluate the learning algorithm A by comparing its output f with f_* .

- If every function f that we treat is represented as a parameter p, we compare p for f and p_* for f_* .
  - Every linear inequation $(\boldsymbol{w}, \boldsymbol{x}) + c > 0$ is represented as a parameter vector $(\boldsymbol{w}, c)$.
  - We evaluate A with comparing $(\boldsymbol{w}, c)$ and $(\boldsymbol{w}_*, c_*)$.

Correctness with Unknown Functions (1)

- Assuming an unknown discriminant function f_* , we could say that the learning algorithm A is correct if the output f of A becomes *nearer* f_* when more data are fed to A.
- Mathematically, consider a infinite sequence of training data sets $(C_0, D_0), (C_1, D_1), (C_2, D_2), \ldots$ such that

$$C_0 \subset C_1 \subset C_2 \subset \ldots \subset C_* \text{ and}$$

$$D_0 \subset D_1 \subset D_2 \subset \ldots \subset D_* .$$

Let f_i be the output of A for C_i and D_i.

Then the algorithm A is correct if $\| f_i - f_* \| \to 0$ for any of such sequences.

Correctness with Unknown Functions (2)

- A similar definition of correctness could be defined:
  If the learning algorithm A is correct if
  A outputs f_* whenever an enough amount of training data are fed to A.
- Mathematically, consider a infinite sequence of training data sets $(C_1, D_1), (C_2, D_2), (C_3, D_3), \ldots$ such that

$$C_1 \subset C_2 \subset C_3 \subset \ldots \subset C_* \text{ and}$$

$$D_1 \subset D_2 \subset D_3 \subset \ldots \subset D_*.$$

Let f_i be the output of A for C_i and D_i.
Then the algorithm A is correct if for each of such sequences, there exists an N such that $\| f_i - f_* \| = 0$ for all $n \geq N$.

Estimation and Learning

- Estimation in statistics means to infer the value of parameters from examples.
- We assume an unknown value of θ.
- The parameter θ affects the distribution of $D(\theta)$, and only finite number of data are coming from the set.
- We expect that, more data from $D(\theta)$, better conjecture θ^ could be obtained.
- The conjecture θ^ is (statistically) consistent if

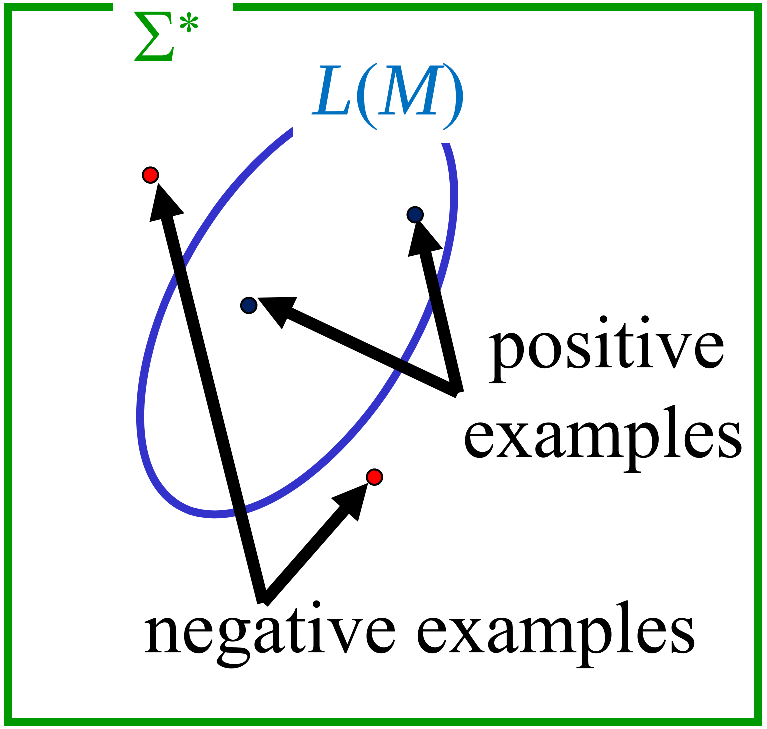
$$\lim_{n\to\infty} \mathrm{E}(\theta\,\hat{}\,) = \theta$$

Correctness of Learning Automata

Examples on $L(M)$

- We assume that, for an unknown automaton M_*,

C_* is a finite set of positive examples on $L(M_*)$ and
D_* is a finite set of negative examples on $L(M_*)$.



$L(M)$: a language accepted by a finite state automaton M

- a positive example on $L(M)$:

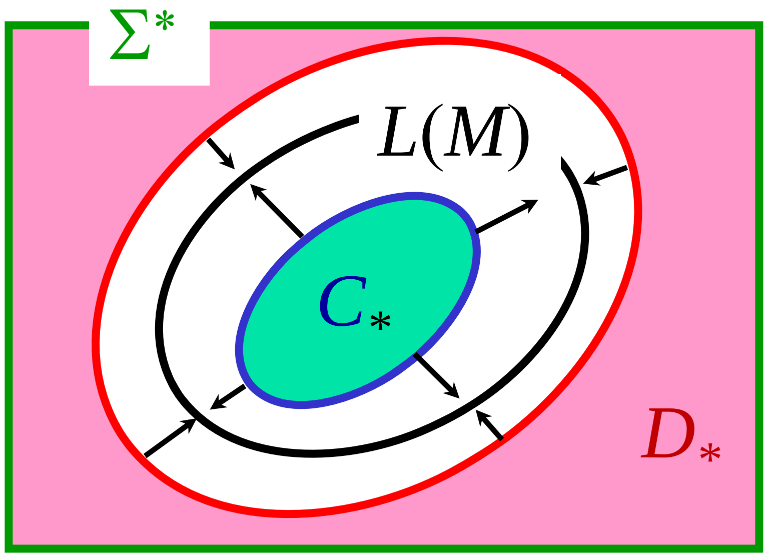
$$< x, +> \text{ for } x \in L(M)$$

- a negative example on $L(M)$:

$$< x, -> \text{ for } x \in \overline{L(M)}$$

Question

- If we give more and more (negative and positive) examples on $L(M_*)$ to an learning algorithm, does it eventually conjecture the unknown M_*?
- We have to give mathematical definitions of
  - giving more and more examples, and
    - or giving examples many enough
  - conjecturing M eventually.



$M^\wedge \rightarrow M$

Assumption

- Without loss of generality, we may assume that learning algorithm takes examples in C_* and D_* one by one.
- In the situation that both C_i and D_i grow, we assume that an infinite sequence σ of strings marked with either + or –, and some truncation of σ corresponds to C_i and D_i.

Example

σ : <ab,+>, <aab,+>, <bbb,–>, <aaab,+>, <abba,–>, | ..

C_i = {ab, aab, aaab},

D_i = {bbb, abba}.

Presentations

Definition A presentation of $L(M)$ is a infinite sequence

$$\sigma: <s_0, p_0>, <s_1, p_1>, <s_2, p_2>, \ldots$$

where $s_i \in \Sigma^*$ and $p_i = +$ or $-$.

- $<s, +>$ is a positive example
- $<s, ->$ is a negative example

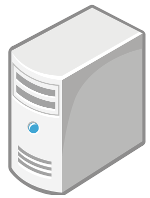
- $\sigma[n] = <s_0, p_0>, <s_1, p_1>, <s_2, p_2>, \ldots, <s_{n-1}, p_{n-1}>$

Definition A presentation σ is complete if

any $x \in L(M)$ appears in σ as a positive example $<x, +>$ at least once and

any $x \in \overline{L(M)}$ appears in σ as a negative example $<x, ->$ at least once.

Identification in the limit [Gold]

 $x_1, x_2, x_3, \ldots$  $M_1, M_2, M_3, \ldots$

- A learning algorithm A EX-identifies $L(M)$ in the limit from complete presentations if
  for any complete presentation $\sigma = x_1, x_2, x_3, \ldots$ of $L(M)$
  and the output sequence $M_1, M_2, M_3, \ldots$ of A, there exists
  N such that for all $n \geq N$ $M_n = M'$ and $L(M') = L(M)$
- A learning algorithm A BC-identifies $L(M)$ in the limit from complete presentations if
  for any complete presentation $\sigma = x_1, x_2, x_3, \ldots$ of $L(M)$
  and the output sequence $M_1, M_2, M_3, \ldots$ of A, there exists
  N such that for all $n \geq N$ ~~$M_n = M'$ and~~ $L(M_n) = L(M)$

A Well-known Result on FA

Theorem For every language $L(M)$ accepted by a finite state automaton M, there exists a unique minimal automaton M' such that $L(M)=L(M')$, where "minimal" means that the number of states in M' is minimal in such automata.

Embedding the Modified Generate-and-Test Algorithm into the Framework

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

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

Input $\sigma = x_1, x_2, \ldots$: presentation (an infinite sequence)

Initialize $k = 0$ /*M_0 as an automaton consisting of one state*/

for $N = 1,2,\ldots$

  $\sigma[N] = x_1, x_2, \ldots, x_N$

  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

On the Generate-and-Test Algorithm

Theorem For any finite state automaton M_* on Σ,

the modified generate-and-test algorithm EX-identifies $L(M_*)$ in the limit from complete presentations.

Proof Let σ be an any complete presentation on $L(M_*)$.

Let M_N be the output of the algorithm for the input $\sigma[N]$.

If $L(M_*) \neq L(M_N)$, then there must be a string $x \in \Sigma^*$

$(x \in L(M_*)$ and $x \notin L(M_N))$ or $(x \notin L(M_*)$ and $x \in L(M_N))$.

Since σ is complete, x must be appears in the sequence with the sign + if $x \in L(M)$ or otherwise with – .

This means that M_N must be replaced with another automaton, at latest, when x appears in σ.

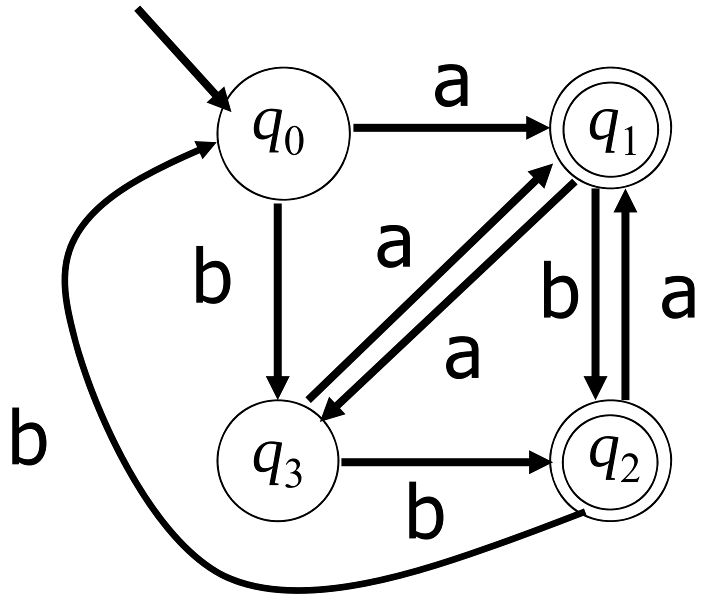
Once the algorithm outputs M_N s.t. $L(M_*) = L(M_N)$, it never changes the output afterwards.

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 $\delta(q_0, x) = q_i$.

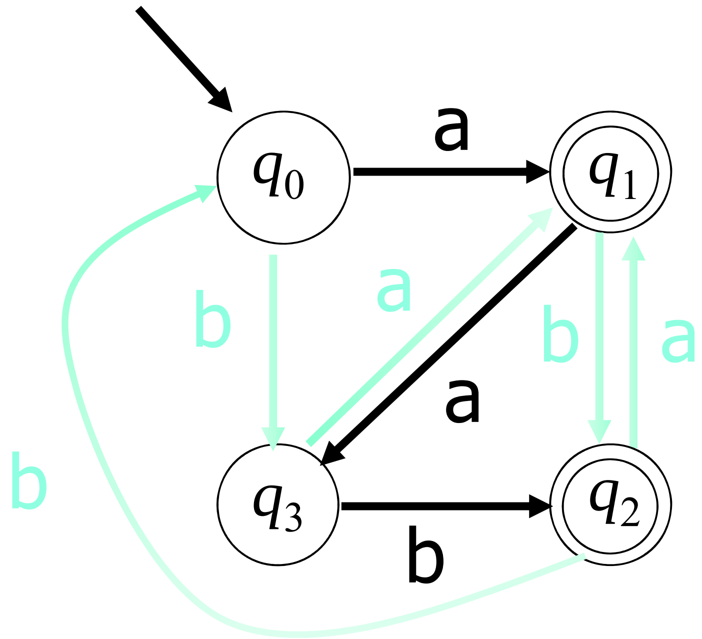
Example Examples of test sets of M are $S_1 = \{\varepsilon$, a, aa, aab$\}$ and $S_2 = \{\varepsilon$, a, ab, b$\}$.



Minimal Test Sets

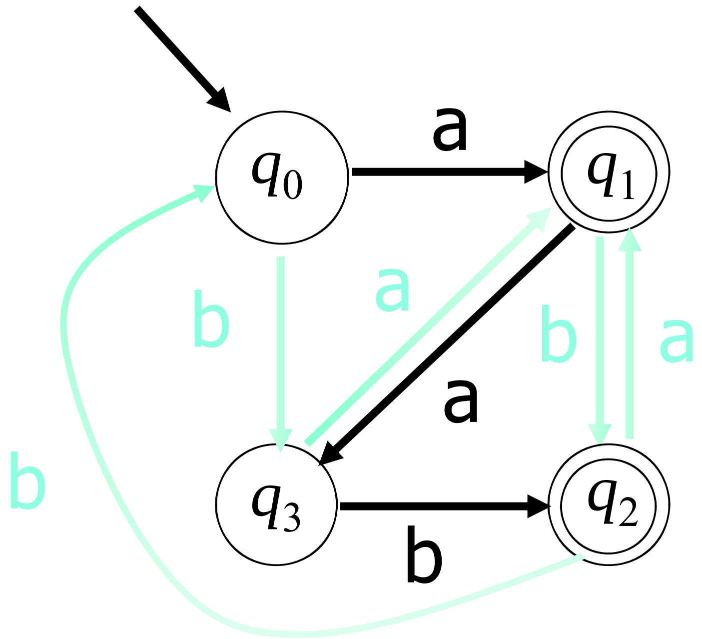
- Intuitively, a test set gives a "skelton" of the finite state automaton.
  - But the set is not sufficient to identify the FA.

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



Prefix closed Test Sets

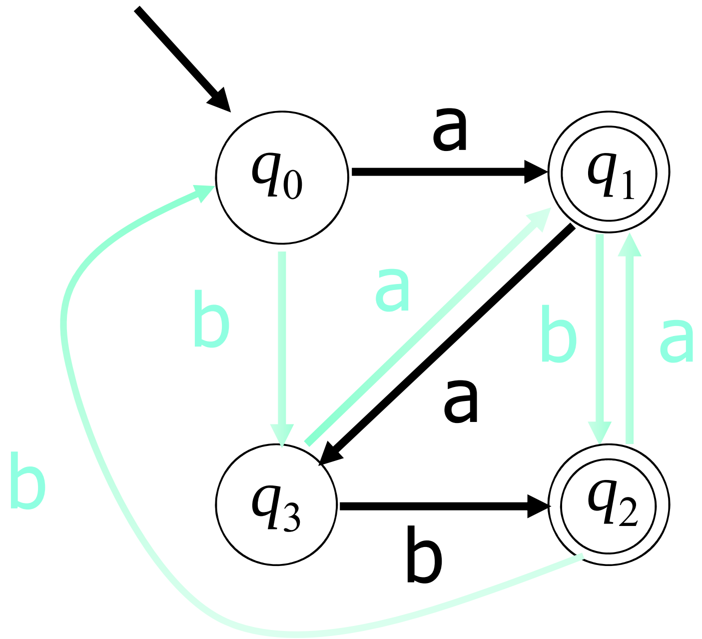
- A set of strings S is prefix closed (suffix closed) if and only if every prefix (resp. suffix) of every member of S is also a member of S.
- Intuitively, a prefix closed minimal test set gives a "skelton" of the finite state automaton.
  - But the set is not sufficient to identify the FA.



Example Both S_1 = {ε, a, aa, aab} and S_2 = {ε, 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.

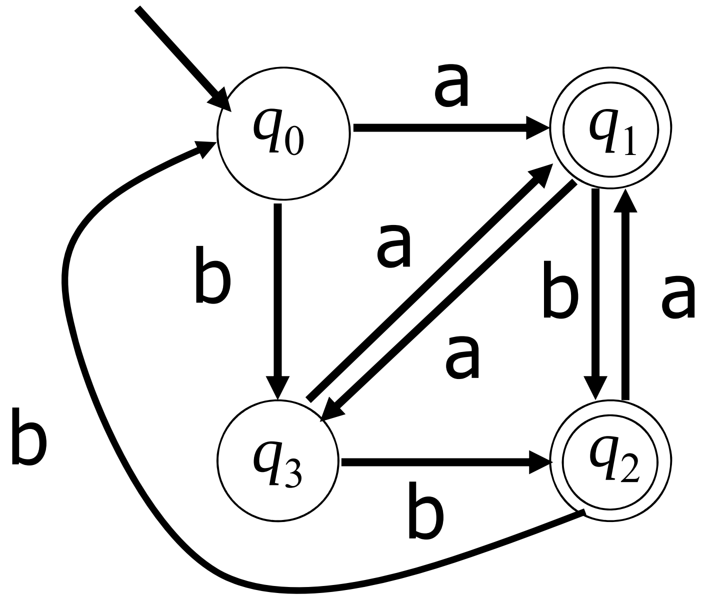


Example Both S_1 = {ε, a, aa, aab} and S_2 = {ε, a, ab, b} are prefix closed.

Fixing an Order

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

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

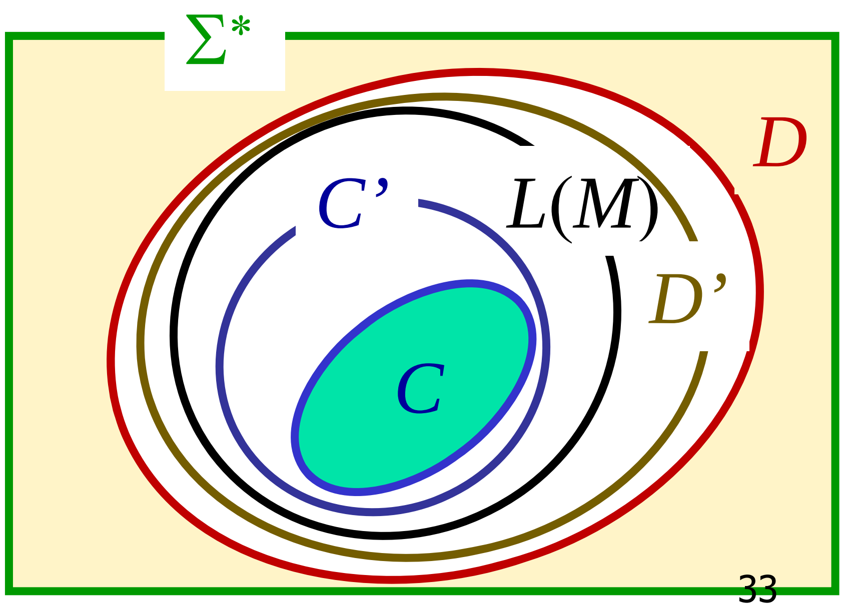


Characteristics Examples

- 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.



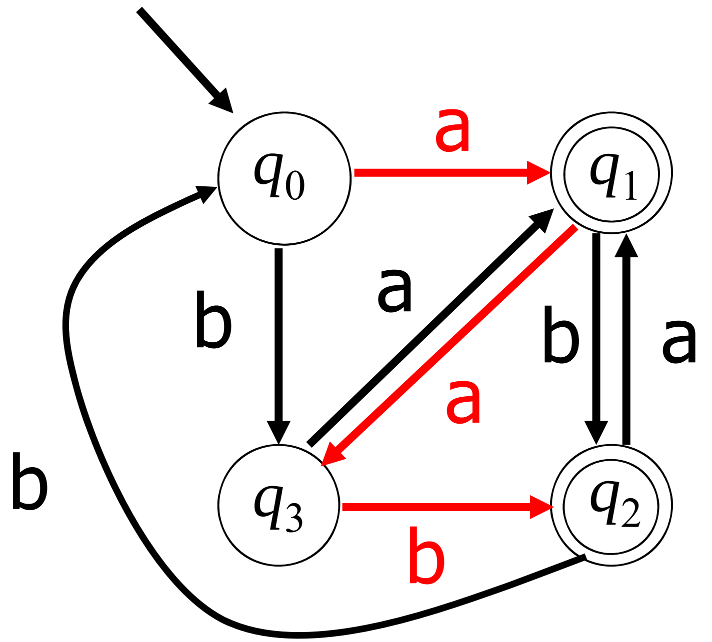
Observation table

- 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	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

Observation table

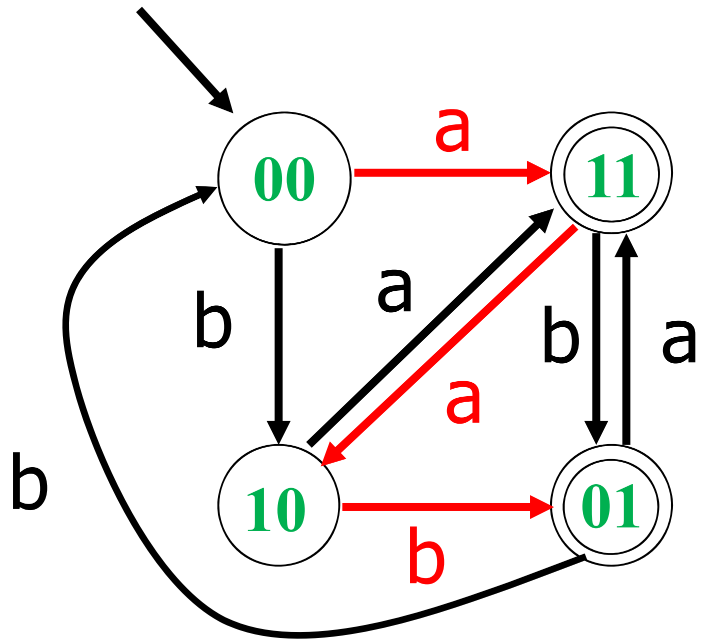
- An observation table (S, E, T) :
  S : a prefix closed set $S \subset \Sigma^*$
  E : a set $E \subset \Sigma^*$
  $T : (S \cup S\,\Sigma)E \rightarrow \{0, 1\}$



		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

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
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

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 := { ε }, 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

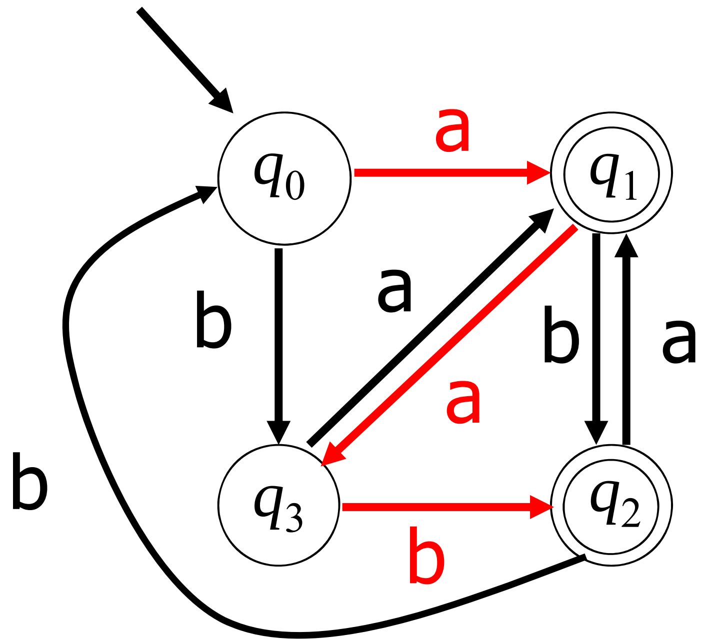
$C = \{ we \mid w \in S \cup S\Sigma, e \in E, \text{ and } T(wc, e) = 1\}$

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

return (C, D);

Example

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



E

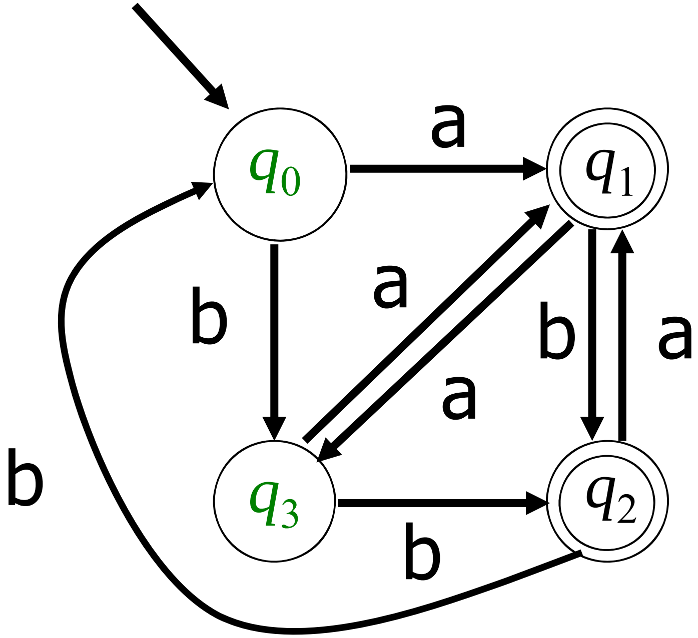
S

$S\,\Sigma$

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

Example

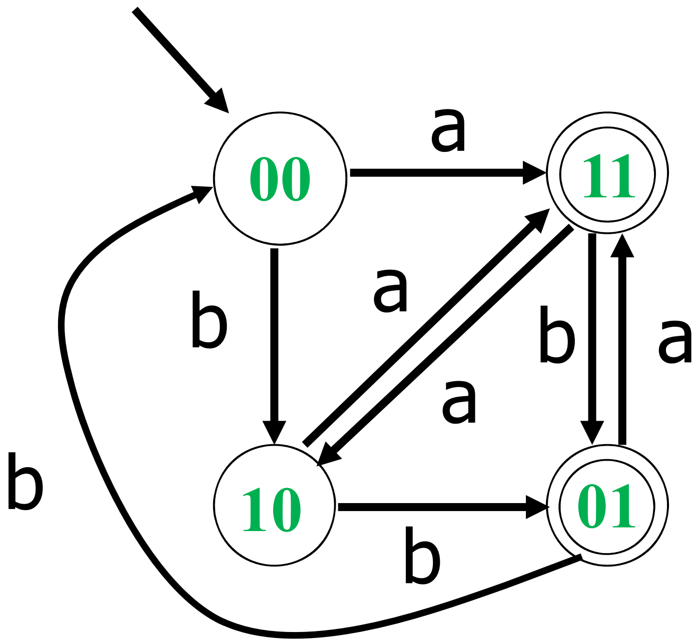
Because $T(\varepsilon, \varepsilon) = T(\text{aa}, \varepsilon)$, check whether or not $T(\text{a}, \varepsilon) = T(\text{aaa}, \varepsilon)$, and
whether or not $T(\text{b}, \varepsilon) = T(\text{aab}, \varepsilon)$.



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

Example

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

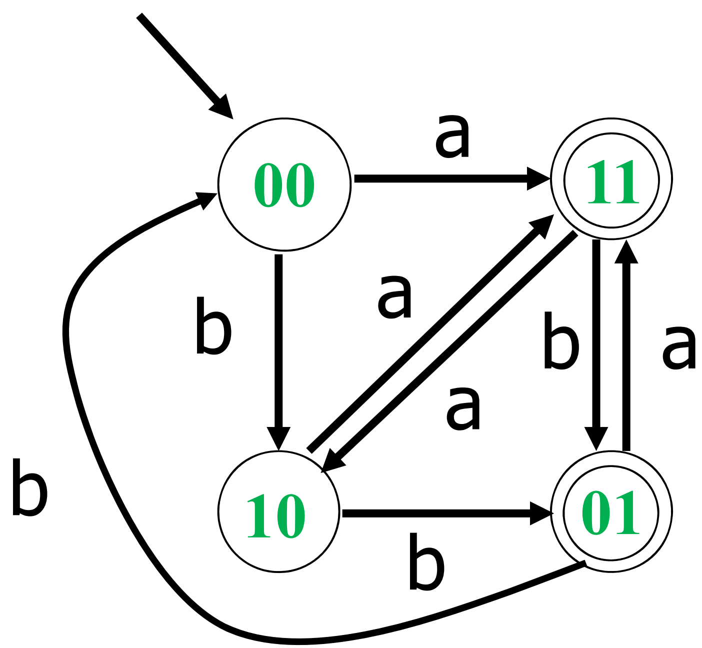


	ε	b
ε	0	0
a	1	1
aa	0	1
aab	1	0
b	0	1
ab	1	0
aaa	1	1
aaba	1	1
aabb	0	0

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}



		E: ε	E: 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

Consistent Table

- An observation table (S, E, T) is consistent if and only if for every pair $w, v \in S$ such that $\text{row}(w) = \text{row}(v)$, $\text{row}(wc) = \text{row}(vc)$ 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.

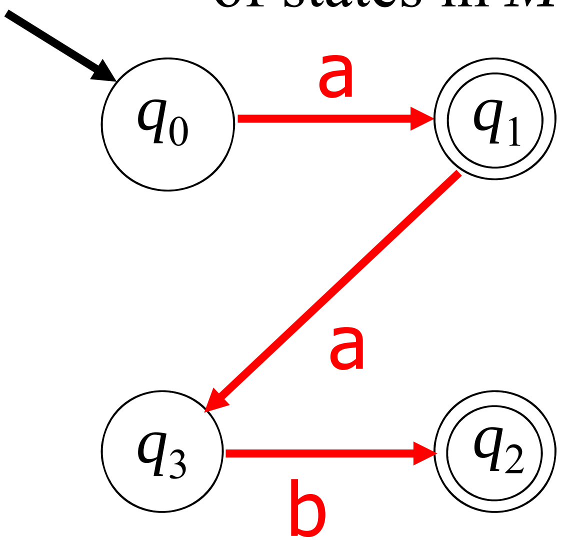
- An equivalence relation R is right invariant iff $x\ R\ y$ implies $xz\ R\ yz$ for all $z \in \Sigma^*$.
- The index of equivalence relation R is the number of equivalence classes.

Why Characteristic Set?

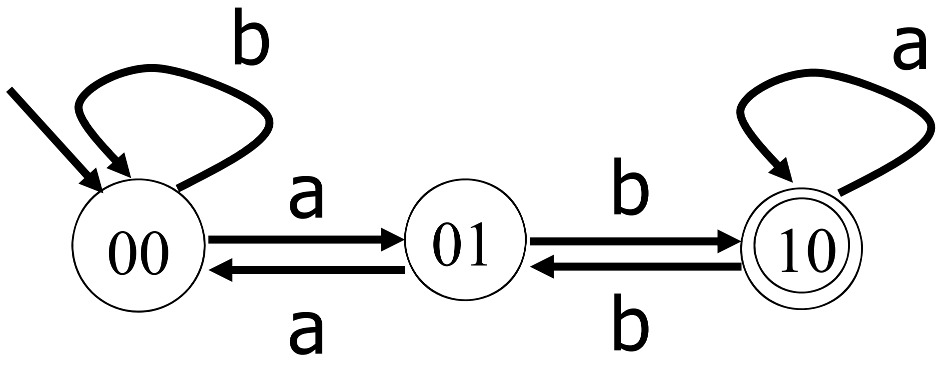
- Let M' be an automaton the number of whose states is minimal and less than that of M. Then it holds that

  $$C \cap \overline{L(M')} \neq \emptyset \text{ or } D \cap L(M') \neq \emptyset.$$

  - Proof : Since S is a test set minimal and prefix closed, we can construct a prefix tree T such that there is a one-to-one and on-to mapping between the set of nodes in T and the set of states in M. If $C \subseteq L(M')$ and $D \subseteq \overline{L(M')}$, then some node in M' must corresponds to more than two nodes in T. However from Myhill-Nerode's theorem, the set E avoids such correspondence.

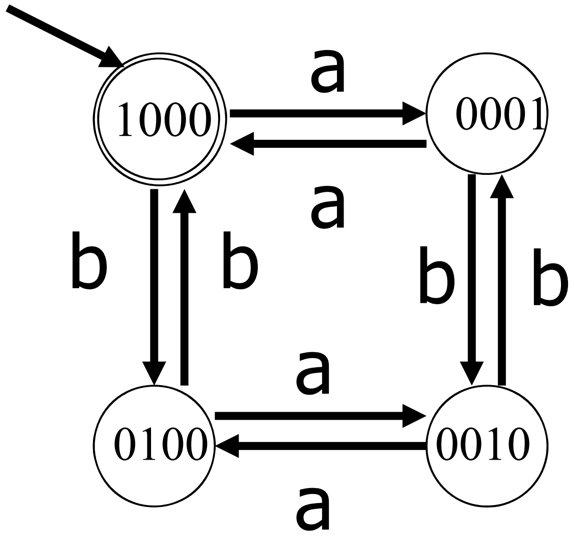


Example 2



	ε	b
ε	0	0
a	0	1
ab	1	0
b	0	0
aa	0	0
aba	1	0
abb	0	1

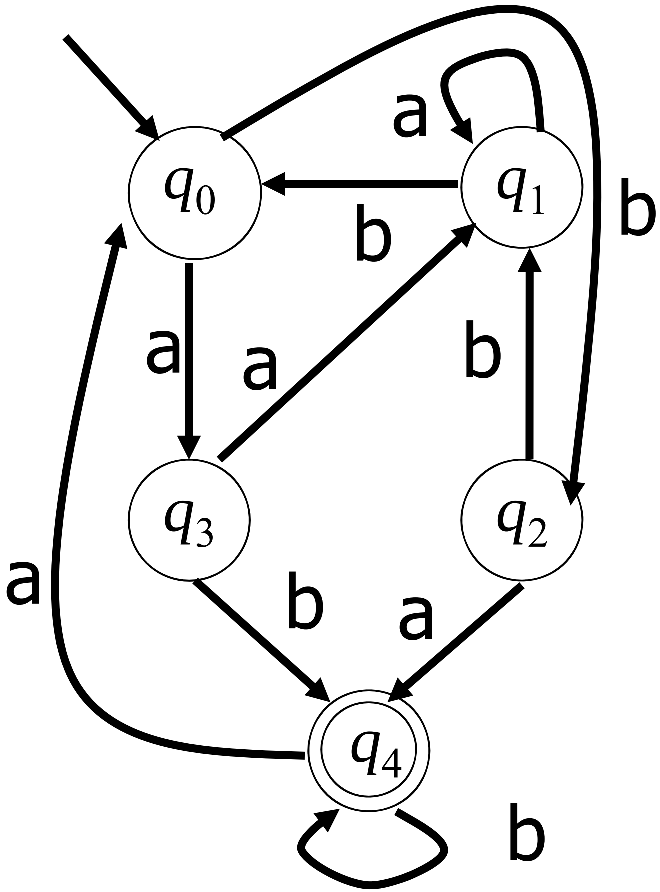
Example 3



E = {ε, b, ab, bab}; S = {ε, a, b, ab}; $S\Sigma$ = {ab, aa, ba, bb, aba, abb}

	ε	b	ab	bab
ε	1	0	0	0
a	0	0	0	1
b	0	1	0	0
ab	0	0	1	0
aa	1	0	0	0
ba	0	0	1	0
bb	1	0	0	0
aba	0	1	0	0
abb	0	0	0	1

Example 4

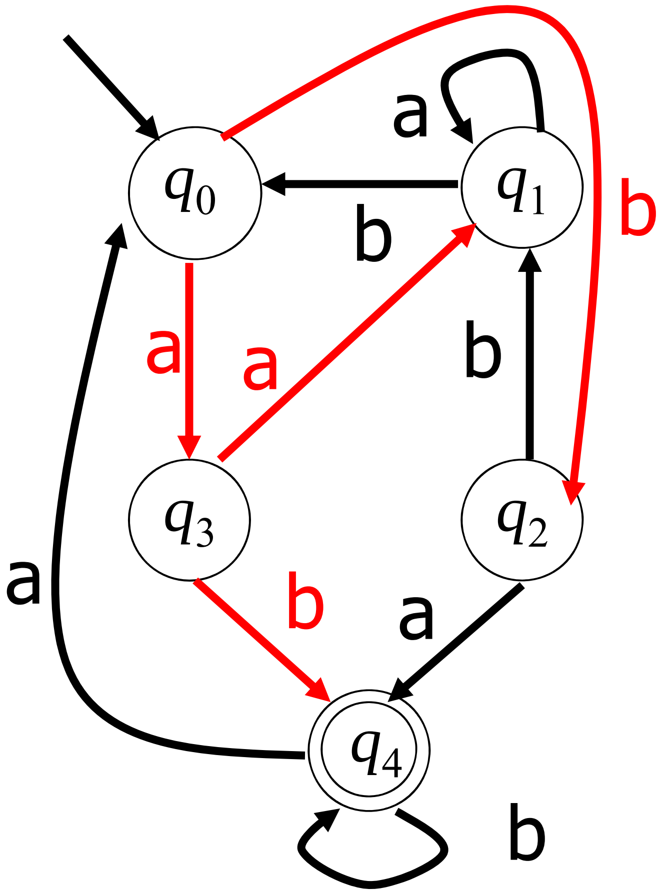


		E		
	ε	b	a	ab
ε	0	0	0	1
a	0	1	0	0
aa	0	0	0	0
ab	1	1	0	0
b	0	0	1	1
ba	1	1	0	0
bb	0	0	0	0
aaa	0	0	0	0
aab	0	0	0	1
aba	0	0	0	1
abb	1	1	0	0

S

$S\,\Sigma$

Example 4



		E			
		ε	b	a	ab
S	ε	0	0	0	1
	a	0	1	0	0
	aa	0	0	0	0
	ab	1	1	0	0
	b	0	0	1	1
$S\,\Sigma$	ba	1	1	0	0
	bb	0	0	0	0
	aaa	0	0	0	0
	aab	0	0	0	1
	aba	0	0	0	1
	abb	1	1	0	0