FSM-Based Testing
Part I

Mohammad Mousavi

Eindhoven University of Technology, The Netherlands

Software Testing, 2011

Outline

Finite State Machines

Finite State Machine

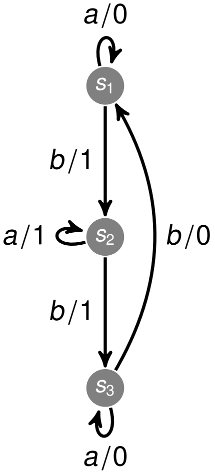
$$M = (I, O, S, \delta, \lambda)$$

with

- I, O, and S finite and non-empty sets of input symbols, output symbols, and states
- state transition function $\delta : S \times I \to S$
- output function $\lambda : S \times I \to O$

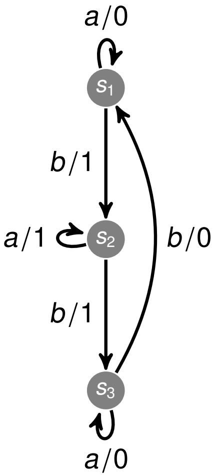
Note that M is deterministic (and complete)

Representations of FSM



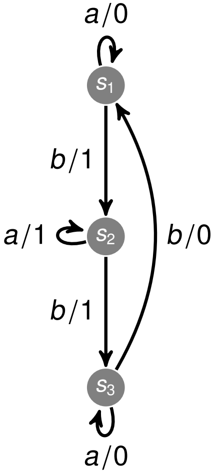
Notations I

$$\delta : S \times I^{*} \rightarrow S$$



Notations I

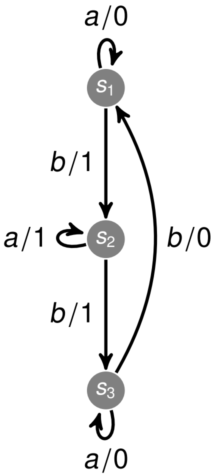
$$\delta : S \times I^* \to S$$
$$\delta(s, \epsilon) = s$$
$$\delta(s, a\ w) = \delta(\delta(s, a), w)$$



Notations I

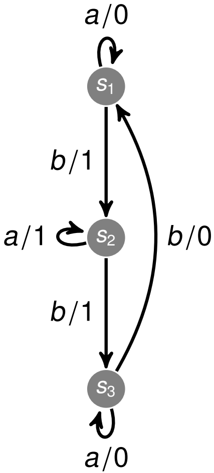
$$\delta : S \times I^* \rightarrow S$$
$$\delta(s, \epsilon) = s$$
$$\delta(s, a\ w) = \delta(\delta(s, a), w)$$

$$\begin{aligned}\delta(s_1, bba) &= \delta(\delta(s_1, b), ba) \\ &= \delta(s_2, ba) \\ &= \delta(\delta(s_2, b), a) \\ &= \delta(s_3, a) \\ &= s_3\end{aligned}$$



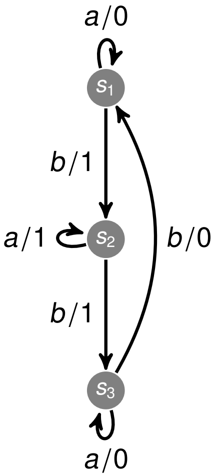
Notations II

$$\lambda : S \times I^* \rightarrow O^*$$



Notations II

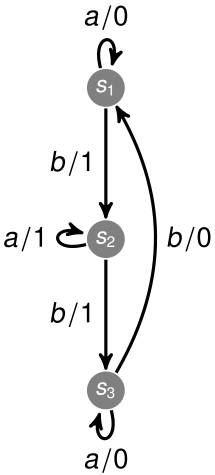
$$\lambda : S \times I^* \rightarrow O^*$$
$$\lambda(s, \epsilon) = \epsilon$$
$$\lambda(s, a\ w) = \lambda(s, a)\ \lambda(\delta(s, a), w)$$



Notations II

$$\lambda : S \times I^* \rightarrow O^*$$
$$\lambda(s, \epsilon) = \epsilon$$
$$\lambda(s, a\ w) = \lambda(s, a)\ \lambda(\delta(s, a), w)$$

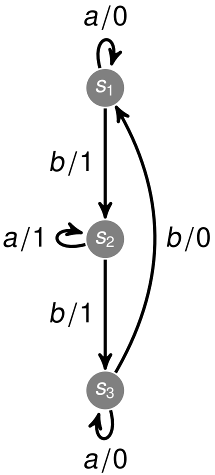
$$\begin{aligned} \lambda(s_1, bba) &= \lambda(s_1, b)\ \lambda(\delta(s_1, b), ba) \\ &= 1\ \lambda(s_2, ba) \\ &= 1\ \lambda(s_2, b)\ \lambda(\delta(s_2, b), a) \\ &= 1\ 1\ \lambda(s_3, a) \\ &= 1\ 1\ 0 \end{aligned}$$



Notations III

$$\delta : 2^S \times I^* \to 2^S$$
$$\delta(Q, x) = \{\delta(s, x) \mid s \in Q\}$$

$$\lambda : 2^S \times I^* \to 2^{O^*}$$
$$\lambda(Q, x) = \{\lambda(s, x) \mid s \in Q\}$$



Initial state uncertainty

$\pi(x)$ for $x \in I^*$: those a partitioning of S, where s_i, s_j are in the same partition iff $\lambda(s_i, x) = \lambda(s_j, x)$.

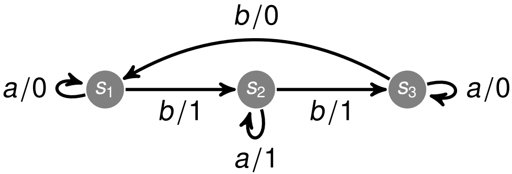
Intuition: by observing output you cannot tell where you were

Initial state uncertainty

$\pi(x)$ for $x \in I^*$: those a partitioning of S, where s_i, s_j are in the same partition iff $\lambda(s_i, x) = \lambda(s_j, x)$.

Intuition: by observing output you cannot tell where you were

Example



$\pi(b) = \{\{s_1, s_2\}, \{s_3\}\}$ $\pi(aa) = \{\{s_1, s_3\}, \{s_2\}\}$

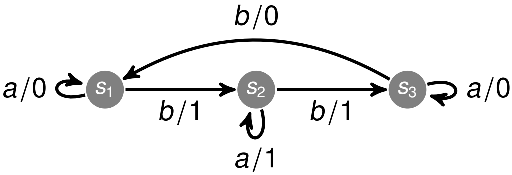
Current state uncertainty

$\sigma(x)$ for $x \in I^*$: a family of sets of states

$$\sigma(x) = \{\delta(B, x) \mid B \in \pi(x)\}$$

Intuition: after applying x you do not know where you are

Example



$\pi(b) = \{\{s_1, s_2\}, \{s_3\}\}$
$\sigma(b) = \{\{s_2, s_3\}, \{s_1\}\}$

$\pi(aa) = \{\{s_1, s_3\}, \{s_2\}\}$
$\sigma(aa) = \{\{s_1, s_3\}, \{s_2\}\}$

Equivalence

Let $M = (I, O, S, \delta, \lambda)$ and $M' = (I, O, S', \delta', \lambda')$

State equivalence for $s, s' \in S$:

$$s \approx s' \quad \doteq \quad \forall_{x \in I^*} \; \lambda(s, x) = \lambda(s', x)$$

Equivalence

Let $M = (I, O, S, \delta, \lambda)$ and $M' = (I, O, S', \delta', \lambda')$

State equivalence for $s, s' \in S$:

$$s \approx s' \quad \doteq \quad \forall_{x \in I^*} \; \lambda(s, x) = \lambda(s', x)$$

State equivalence: for $s \in S$ and $s' \in S'$

$$s \approx s' \quad \doteq \quad \forall_{x \in I^*} \; \lambda(s, x) = \lambda'(s', x)$$

Equivalence

Let $M = (I, O, S, \delta, \lambda)$ and $M' = (I, O, S', \delta', \lambda')$

State equivalence for $s, s' \in S$:

$$s \approx s' \quad \doteq \quad \forall_{x \in I^*} \; \lambda(s, x) = \lambda(s', x)$$

State equivalence: for $s \in S$ and $s' \in S'$

$$s \approx s' \quad \doteq \quad \forall_{x \in I^*} \; \lambda(s, x) = \lambda'(s', x)$$

Machine equivalence

$$M \approx M' \quad \doteq \quad \forall_{s \in S} \exists_{s' \in S'} \; s \approx s' \wedge \forall_{s' \in S'} \exists_{s \in S} \; s' \approx s$$

Minimization of FSM

- Let $[s]_{/\approx} = \{s' \in S \mid s \approx s'\}$.
- Define $S_{\min} = \{[s]_{/\approx} \mid s \in S\}$ to be the set of equivalence classes of S with respect to state equivalence.
- Define $\lambda_{\min}([s]_{/\approx}, a) = \lambda(s, a)$ for all s and a.
- Define $\delta_{\min}([s]_{/\approx}, a) = [\delta(s, a)]_{/\approx}$.

Minimization of FSM

- Let $[s]_{/\approx} = \{s' \in S \mid s \approx s'\}$.
- Define $S_{\mathrm{min}} = \{[s]_{/\approx} \mid s \in S\}$ to be the set of equivalence classes of S with respect to state equivalence.
- Define $\lambda_{\mathrm{min}}([s]_{/\approx}, a) = \lambda(s, a)$ for all s and a.
- Define $\delta_{\mathrm{min}}([s]_{/\approx}, a) = [\delta(s, a)]_{/\approx}$.

Property

Let $M = (I, O, S, \lambda, \delta)$ and $M_{\mathrm{min}} = (I, O, S_{\mathrm{min}}, \lambda_{\mathrm{min}}, \delta_{\mathrm{min}})$. Then $M \approx M_{\mathrm{min}}$.

Minimization of FSM

- Let $[s]_{/\approx} = \{s' \in S \mid s \approx s'\}$.
- Define $S_{\min} = \{[s]_{/\approx} \mid s \in S\}$ to be the set of equivalence classes of S with respect to state equivalence.
- Define $\lambda_{\min}([s]_{/\approx}, a) = \lambda(s, a)$ for all s and a.
- Define $\delta_{\min}([s]_{/\approx}, a) = [\delta(s, a)]_{/\approx}$.

Property

Let $M = (I, O, S, \lambda, \delta)$ and $M_{\min} = (I, O, S_{\min}, \lambda_{\min}, \delta_{\min})$. Then $M \approx M_{\min}$.

Property

Let $M = (I, O, S, \lambda, \delta)$ and $M_{\min} = (I, O, S_{\min}, \lambda_{\min}, \delta_{\min})$. Then $M_{\min}$ is (one of) the smallest FSMs M' such that $M \approx M'$.

Outline

Finite State Machines

Testing problems

Homing and synchronizing sequences

State identification

State verification

Five Fundamental Testing Problems

A test is a sequence of input symbols

1. Homing/distinguishing sequences: Given M, determine the state after a test

Five Fundamental Testing Problems

A test is a sequence of input symbols

1. Homing/distinguishing sequences: Given M, determine the state after a test
2. State identification: Given M, identify the unknown initial state

Five Fundamental Testing Problems

A test is a sequence of input symbols

1. Homing/distinguishing sequences: Given M, determine the state after a test
2. State identification: Given M, identify the unknown initial state
3. State verification: Given M and a state s, verify that M is in state s

Five Fundamental Testing Problems

A test is a sequence of input symbols

1. Homing/distinguishing sequences: Given M, determine the state after a test
2. State identification: Given M, identify the unknown initial state
3. State verification: Given M and a state s, verify that M is in state s
4. Conformance testing: Given black-box M and FSM A (specification), determine whether M is equivalent to A

Five Fundamental Testing Problems

A test is a sequence of input symbols

1. Homing/distinguishing sequences: Given M, determine the state after a test
2. State identification: Given M, identify the unknown initial state
3. State verification: Given M and a state s, verify that M is in state s
4. Conformance testing: Given black-box M and FSM A (specification), determine whether M is equivalent to A
5. Machine identification: Identify unknown black-box machine M

Outline

Homing sequences

Problem: Given a FSM, we do not know which state it is in
Solution: Perform a test, observe output sequence and determine the final state of the machine

Homing sequences

Problem: Given a FSM, we do not know which state it is in
Solution: Perform a test, observe output sequence and determine the final state of the machine

- Reduced FSM always has a homing sequence
- FSM that is not reduced may not have a homing sequence

Homing sequences

Problem: Given a FSM, we do not know which state it is in
Solution: Perform a test, observe output sequence and determine the final state of the machine

- Reduced FSM always has a homing sequence
- FSM that is not reduced may not have a homing sequence

Property

x is a homing sequence if and only if all blocks in current state uncertainty $\sigma(x)$ are singletons

Determining a homing sequence

Algorithm: (for reduced machine)

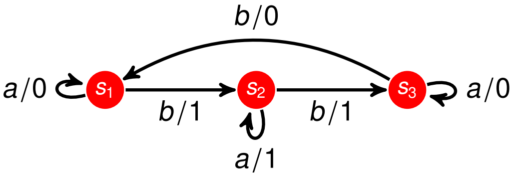
1. let $x = \epsilon$,

Determining a homing sequence

Algorithm: (for reduced machine)

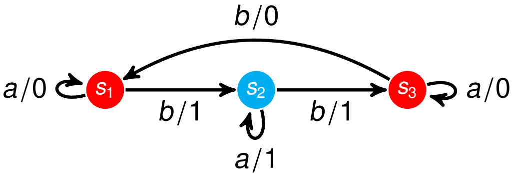
1. let $x = \epsilon$,
2. while there exists a $B \in \sigma(x)$ such that $B > 1$
   2.1 take two states $s, s' \in B$ (with $s \neq s'$)
   2.2 find a sequence y, separating s and s'
   i.e., $\lambda(s, y) \neq \lambda(s', y)$
   2.3 x := x y

Example



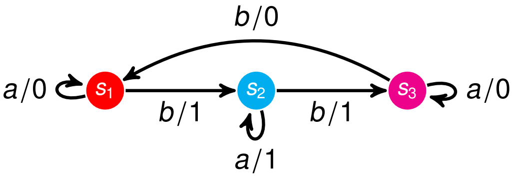
Partition of S: $\{\{s_1, s_2, s_3\}\}$
Take $B = \{s_1, s_2, s_3\}$
Take: s_1 and s_2
Separating sequence: a
Output sequences: $\lambda(s_1, a) = \lambda(s_3, a) = 0$ and $\lambda(s_2, a) = 1$
New partition: $\{\{s_1, s_3\}, \ \{s_2\}\}$

Example



Partition of S: $\{\{s_1, s_3\}, \{s_2\}\}$
Take $B = \{s_1, s_3\}$
Take: s_1 and s_3.
Separating sequence: b
Output sequences: $\lambda(s_1, b) = 1$ and $\lambda(s_3, b) = 0$
New partition: $\{\{s_1\}, \{s_3\}, \{s_2\}\}$

Example



Partition of S: $\{\{s_1\}, \{s_3\}, \{s_2\}\}$
Homing sequence: $a\ b$

- Length of homing sequence: $(|S| - 1)(|S| - 1) = (|S| - 1)^2$
- Finding shortest homing sequence: NP-hard
  Look up shortest homing sequence in successor tree.
  Consider input sequence associated with some node with discrete partition of S

Synchronizing sequence

A sequence x leading to the same final state regardless the initial state and output:

$$x \text{ is synchronizing} \quad \doteq \quad \forall_{s,s' \in S}\ \delta(s,x) = \delta(s',x)$$

Synchronizing sequence

A sequence x leading to the same final state regardless the initial state and output:

$$x \ \text{ is synchronizing} \quad \doteq \quad \forall_{s,s' \in S}\ \delta(s,x) = \delta(s',x)$$

A synchronizing is a homing sequence (not the other way around!)

Existence of a synchronizing sequence

Construct graph with

- nodes $\{\{s, s'\} \mid s, s' \in S\}$
- edge from $\{s, s'\}$ to $\{t, t'\}$ with label a if there are transitions from s to t and from s' to t' with input symbol a

Existence of a synchronizing sequence

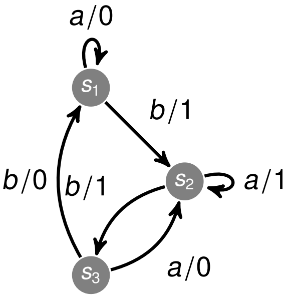
Construct graph with

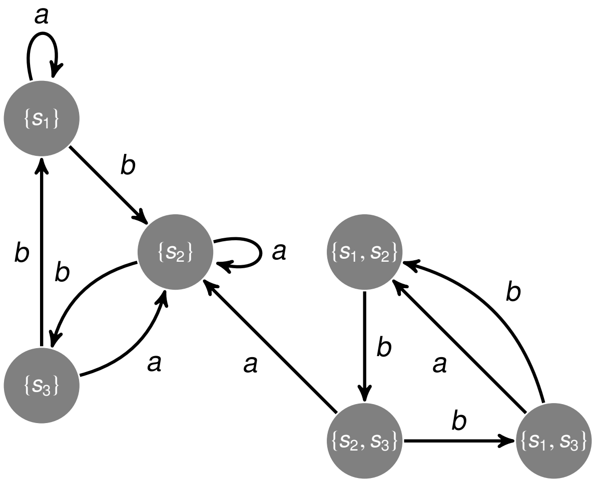
- nodes $\{\{s, s'\} \mid s, s' \in S\}$
- edge from $\{s, s'\}$ to $\{t, t'\}$ with label a if there are transitions from s to t and from s' to t' with input symbol a

Property

An FSM has a synchronizing sequence iff for each $\{s, s'\}$ in the constructed graph (with $s \neq s'$) there is a path to some $\{t\}$.

Example





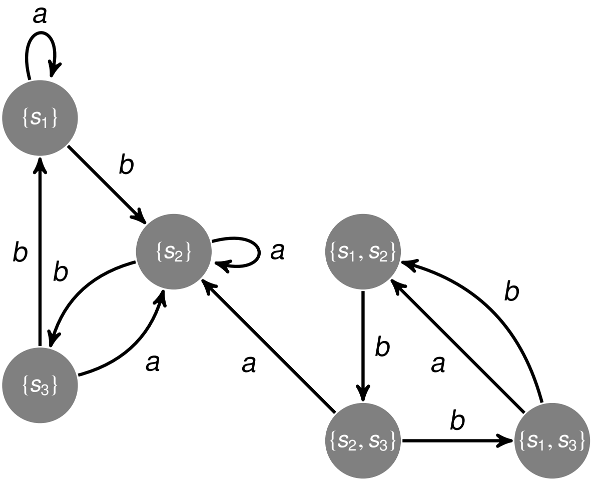
Constructing a synchronizing sequence

1. let $x = \epsilon$,

Constructing a synchronizing sequence

1. let $x = \epsilon$,
2. while $| \delta(S, x) |> 1$
   2.1 take two states $s, s' \in \delta(S, x)$ (with $s \neq s'$)
   2.2 find a sequence y, merging s and s'
   i.e., $\delta(s, y) = \delta(s', y)$
   (it may or may not exist)
   2.3 x := x y

Example



- $S = \{s_1, s_2, s_3\}$. Take states s_2 and s_3. Then $\{s_2, s_3\} \xrightarrow{a} \{s_2\}$. Then $S_1 = \delta(S, a) = \{s_1, s_2\}$
- $S_1 = \{s_1, s_2\}$. Take states s_1 and s_2. Then $\{s_1, s_2\} \xrightarrow{ba} \{s_2\}$. Then $S_2 = \delta(S_1, ba) = \{s_2\}$.
- So *aba* is a synchronizing sequence

Using successor tree for synchronization sequence

Shortest synchronizing sequence can be found from successor tree. Label node reached by input sequence x with $\delta(S, x)$. Look for node with singleton label closest to root.

Outline

Finite State Machines

Testing problems

Homing and synchronizing sequences

State identification

State verification

State identification

Problem: Given a FSM, can we determine the initial state of the FSM?

An input sequence that solves this problem is a distinguishing sequence.

- Preset distinguishing sequences: input sequence is fixed
- Adaptive distinguishing sequences: decision tree (next input symbol depends on onserved outputs)

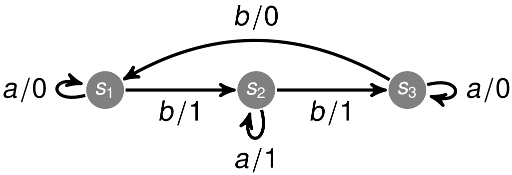
Preset distinguishing sequence

A preset distinguishing sequence for a machine is an input sequence x such that the output sequence in response to x is different for any pair of different states.

$$\forall_{s,s' \in S} \ \lambda(s,x) = \lambda(s',x) \ \Rightarrow \ s = s'$$

- FSM that is not minimal cannot have a preset distinguishing sequence since equivalent states cannot be distinguished from each other by tests

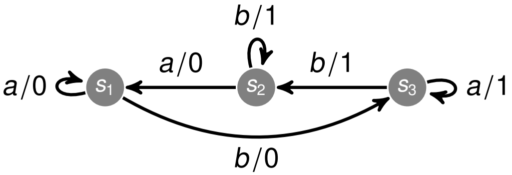
Example



Preset distinguishing sequences:

- not a a since $\lambda(s_1, aa) = \lambda(s_3, aa) = 00$
- a b since $\lambda(s_1, ab) = 01$, $\lambda(s_2, ab) = 11$ and $\lambda(s_3, ab) = 00$
- b a since $\lambda(s_1, ba) = 11$, $\lambda(s_2, ba) = 10$ and $\lambda(s_3, ba) = 00$
- b b since $\lambda(s_1, bb) = 11$, $\lambda(s_2, bb) = 10$ and $\lambda(s_3, bb) = 01$

Non-existence of preset distinguishing sequence



- distinguishing sequence cannot start with a because then s_1 and s_2 are not distinguishable

$$\lambda(s_1, aw) = 0\ \lambda(s_1, w) = \lambda(s_2, aw)$$

- distinguishing sequence cannot start with b because then s_2 and s_3 are not distinguishable

$$\lambda(s_2, bw) = 1\ \lambda(s_2, w) = \lambda(s_3, bw)$$

Complexity

- Existence of preset distinguishing sequence: PSPACE-complete
- Length of preset distinguishing: exponential

Adaptive distinguishing sequence

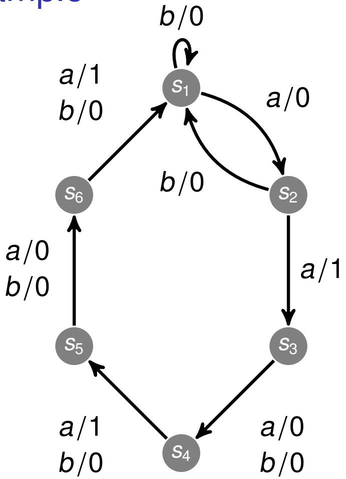
An adaptive distinguishing sequence for a machine is a rooted tree T with exactly $|S|$ leaves

- internal nodes are labeled with input symbols
- leaves are labeled with states
- edges are labeled with output symbols

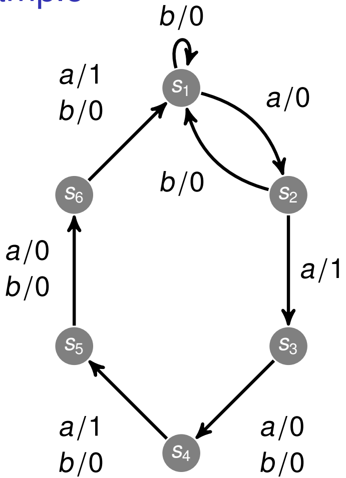
such that

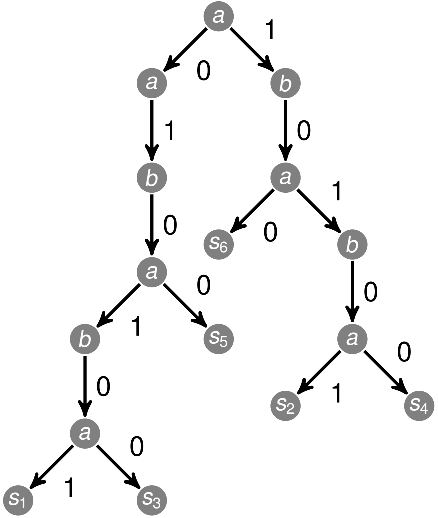
- for each node, the labels of the outgoing edges are different
- for each leaf, if x and y are input and output sequence on path from root to the leaf and the leaf is labeled by state s, then $\lambda(s, x) = y$

Example

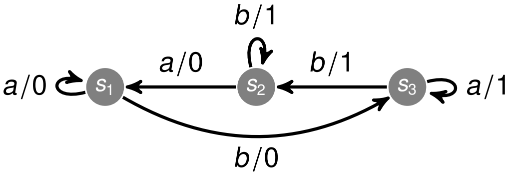


Example


b/0
a/1
b/0
s_1
a/0
b/0
s_6
s_2
a/0
b/0
a/1
s_5
s_3
a/1
b/0
s_4
a/0
b/0


a
1
0
a
b
1
0
b
a
1
0
0
s_6
b
a
0
0
1
b
s_5
a
0
0
1
a
0
s_2
s_4
1
s_1
s_3

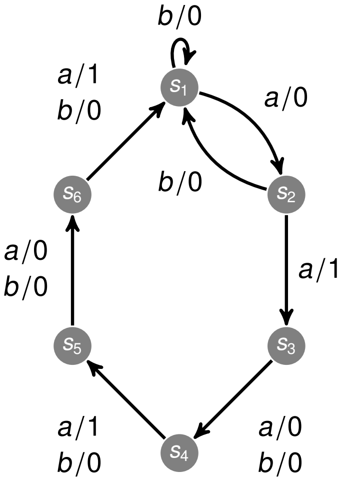
Non-existence of adaptive distinguishing sequence



- distinguishing sequence cannot start with a because then s_1 and s_2 are not distinguishable since from both s_1 is reached with output 0
- distinguishing sequence cannot start with b because then s_2 and s_3 are not distinguishable since from both s_2 is reached with output 1

Preset versus adaptive distinguishing sequences

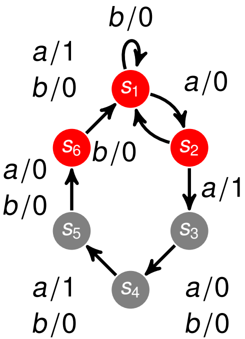
- If FSM has preset distinguishing sequence, then it has an adaptive distinguishing sequence
- An FSM with an adaptive distinguishing sequence does not have to have a preset distinguishing sequence



Existence of adaptive distinguishing sequence

An input *a* is valid for a set *C* of states if it does not merge any two states *s* and *s′* from *C* without distinguishing them, i.e.,

$$\forall_{s,s' \in C} \;\; \lambda(s,a) \neq \lambda(s',a) \vee \delta(s,a) \neq \delta(s',a)$$



- Input symbol *b* is not valid for set of states *S*
- Input symbol *a* is valid for set of states *S*

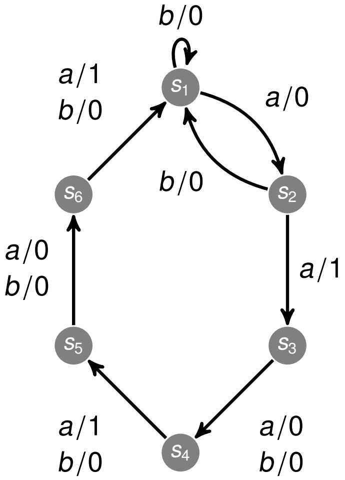
Algorithm

1. Start with partition π of S with only one block $\{S\}$.
2. While there is a block $B \in \pi$ with $|B| > 1$,
   2.1 Take a valid input symbol $a \in I$ for B such that two states $s, s' \in B$ $(s \neq s')$, $\lambda(s, a) \neq \lambda(s', a)$ or move to states in different blocks of π,
   2.2 refine the partition π by replacing block B by a set of new blocks, where two states in B are assigned to the same block in the new partition iff they produce the same output on a and move to the same block in π.

Property

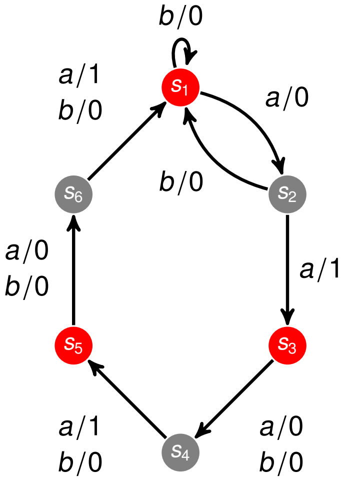
A FSM has an adaptive distinguishing sequence iff the final partition is the discrete partition.

Example



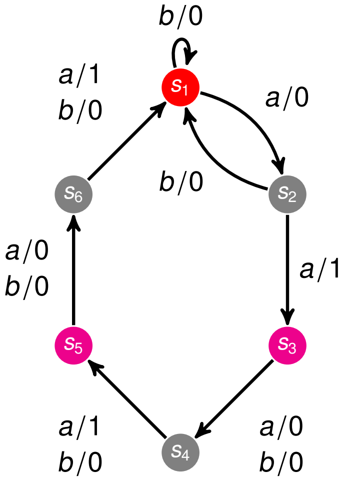
- Initial partition $\pi = \{S\}$
- Input symbol b is not valid
- Input symbol a is valid
- New partition:
  $\pi = \{\{s_1, s_3, s_5\}, \{s_2, s_4, s_6\}\}$

Example



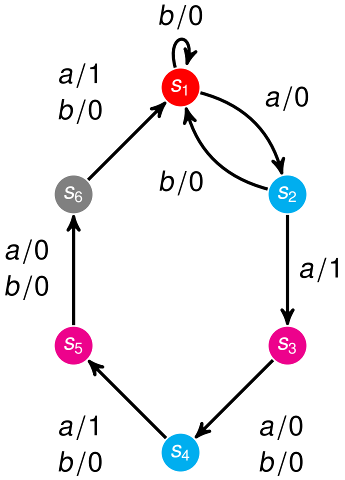
- Initial partition $\pi = \{\{s_1, s_3, s_5\}, \{s_2, s_4, s_6\}\}$
- Input symbol b is valid for $\{s_1, s_3, s_5\}$
- New partition: $\pi = \{\{s_1\}, \{s_3, s_5\}, \{s_2, s_4, s_6\}\}$

Example



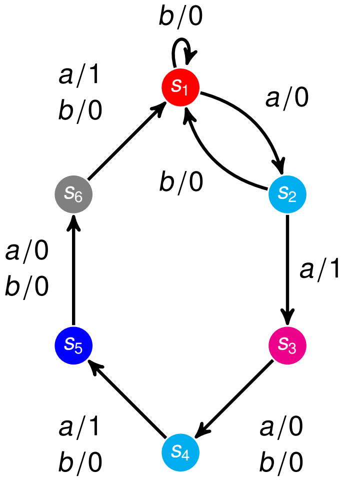
- Initial partition
  $\pi = \{\{s_1\}, \{s_3, s_5\}, \{s_2, s_4, s_6\}\}$
- Input symbol a is valid for $\{s_2, s_4, s_6\}$
- New partition:
  $\pi = \{\{s_1\}, \{s_3, s_5\}, \{s_2, s_4\}, \{s_6\}\}$

Example



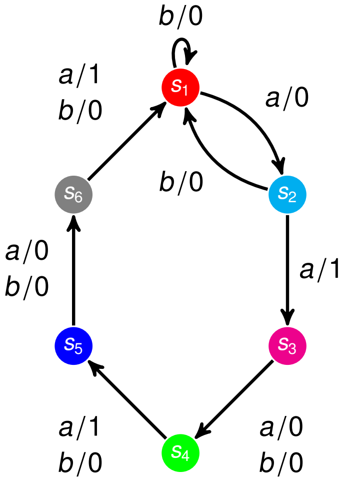
- Initial partition $\pi = \{\{s_1\}, \{s_3, s_5\}, \{s_2, s_4\}, \{s_6\}\}$
- Input symbol b is valid for $\{s_3, s_5\}$
- New partition: $\pi = \{\{s_1\}, \{s_3\}, \{s_5\}, \{s_2, s_4\}, \{s_6\}\}$

Example



- Initial partition $\pi = \{\{s_1\}, \{s_3\}, \{s_5\}, \{s_2, s_4\}, \{s_6\}\}$
- Input symbol a is valid for $\{s_2, s_4\}$
- New partition: $\pi = \{\{s_1\}, \{s_3\}, \{s_5\}, \{s_2\}, \{s_4\}, \{s_6\}\}$
- Thus FSM has an adaptive distinguishing sequence

Example



- Initial partition $\pi = \{\{s_1\}, \{s_3\}, \{s_5\}, \{s_2\}, \{s_4\}, \{s_6\}\}$
- Thus FSM has an adaptive distinguishing sequence

Construction of adaptive distinguishing sequence

1. split conservatively $\Rrightarrow$ construct splitting tree
2. order of splitting (all blocks of largest cardinality simultaneously) $\Rrightarrow$ Construct adaptive distinguishing sequence from splitting tree

See *Principles and Methods of Testing Finite State Machines – A Survey* by D. Lee and M. Yannakakis for details.

Outline

State verification

Problem: Given a FSM an a state s, can we verify by testing that s is the the initial state of the FSM?

This is possible if and only if the FSM has an UIO sequence

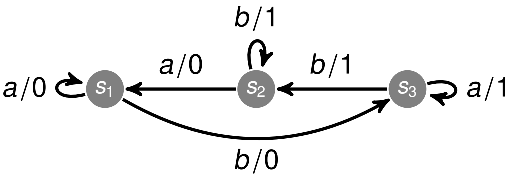
A Unique Input/Output (UIO) sequence of a state s is an input sequence x such that the output produced in response to x from any state other than s is different than from s

$$\forall_{s' \in S} \;\; \lambda(s, x) = \lambda(s', x) \;\Rightarrow\; s = s'$$

Property

x is a UIO sequence for state s if and only if $\{s\} \in \pi(x)$

Example



State s_1 has UIO sequence b
State s_2 does not have a UIO sequence
State s_3 has UIO sequence a

Relationship with state identification

If FSM has (preset or) adaptive distinguishing sequence, then all states have UIO sequences