Principles and Methods of Testing Finite State Machines – A Survey*

- Paper by David Lee and Mihalis Yannakakis
  - Presentation by Kuo-Wen Lo

*D. Lee, M. Yannakakis. Principles and Methods of Testing Finite State Machines – A Survey. Proceedings of IEEE, vol. 84, no. 8, August 1996.

Introduction

- Finite State Machines
  - Demand for system reliability motivates research into problems of testing finite state machines to ensure their correct function, and discover aspects of their behavior.
- This is a survey that covers basic problems of testing finite state machines and present the general principles and methods.

Background

- A FSM is defined as:

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

  I = *set of input symbols*

  O = *set of output symbols*

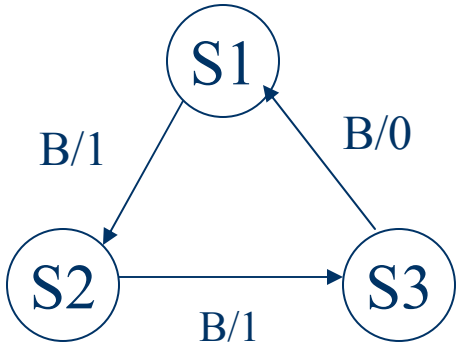
  S = *set of states*

  δ = *state transition function*

  λ = *output function*
- Number of state, input, out: n = |S|, p = |I|, q = |O|
- Let x be equal to a sequence of inputs = a_1, …, a_k, then $\delta(S_1, x)$ take the machine from initial state of S_1, and successively through $\delta(S_i, a_i)$ to a state S_{k+1}.
- Same for the output function λ .

Background

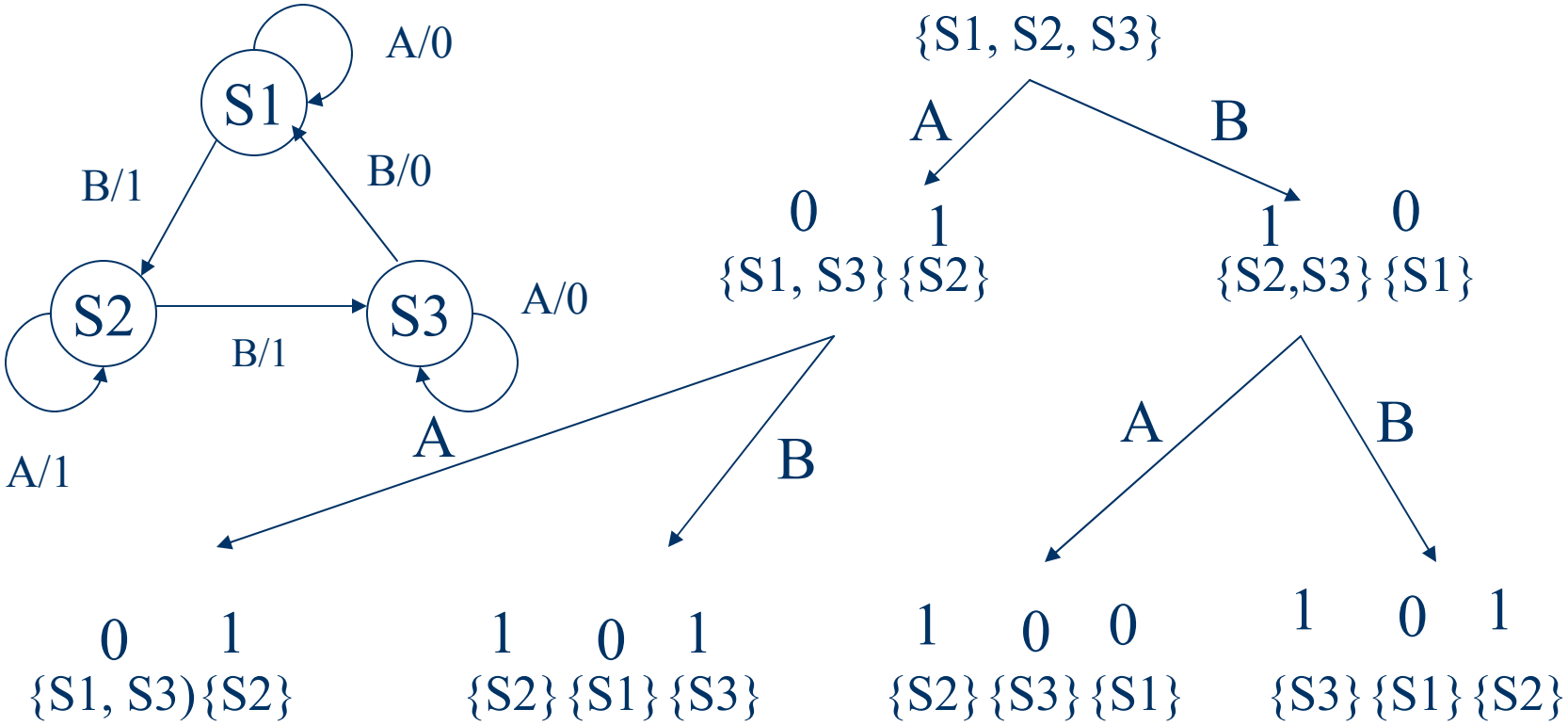
- State uncertainty: Resulting set of states after partitioning base on the input.
  - Initial state uncertainty
  - Current state uncertainty



- Ex: In an FSM, we give an input B. If we see a one, the machine was initially in s1 or s2, and current state is s2 or s3. Therefore, initial state uncertainty of input B is {{s1, s2}, {s3}}, current state of uncertainty is {{s2, s3}, {s1}}.

Background

- Successor Trees



Five fundamental Problems

Tests are done by applying a sequence of input symbols into the machine, observe the output, and determine information

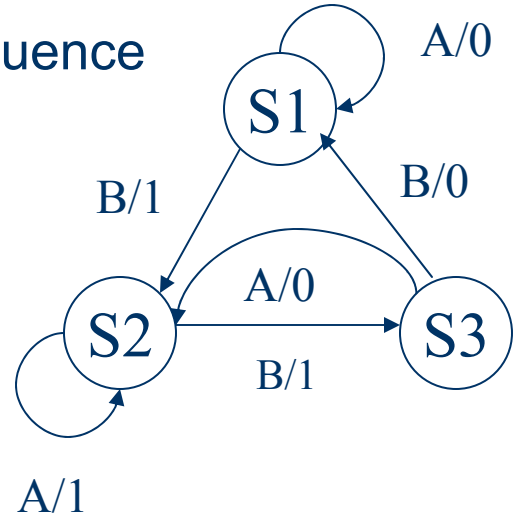
- Determining the final state after the test
- Identify the unknown initial state
- The machine is suppose to be in some initial state. Verify that it is indded in that state.
- Given complete description of another machine A, determine whether the current machine is equivalent to A.
- Identify the unknown machine A.

We're going to focus on the following two main questions:

- Existence – Is there a test sequence that solves the problem?
- If there is, how hard is it to determine whether a sequence exist? How do you find/construct the sequence?

Problem #1: Determining the final state after the test

- Solution: homing/synchronizing sequence
  - Homing sequence: An input sequence such that after it is entered, by observing the output, we know what the final state of the machine is in.
  - Synchronization sequence: An input sequence such that after it is entered, the machine is in a specific final state.
- All reduce machine has a homing sequence
- Not all machines has a synchronizing sequence
- Ex: homing sequence,
  - *BA, 00 – S1, 11 – S2, 10 – S3*
- Ex: synchronizing sequence,
  - ABA – S2

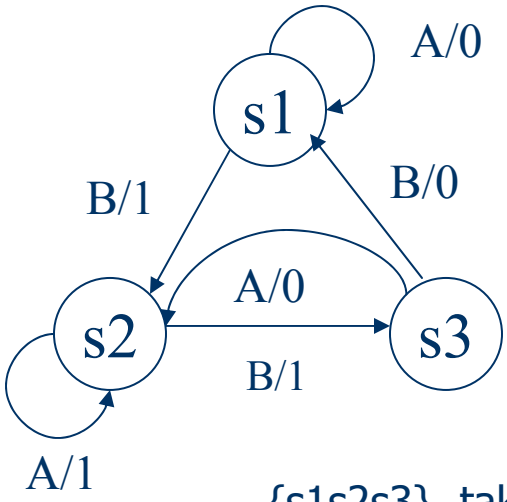


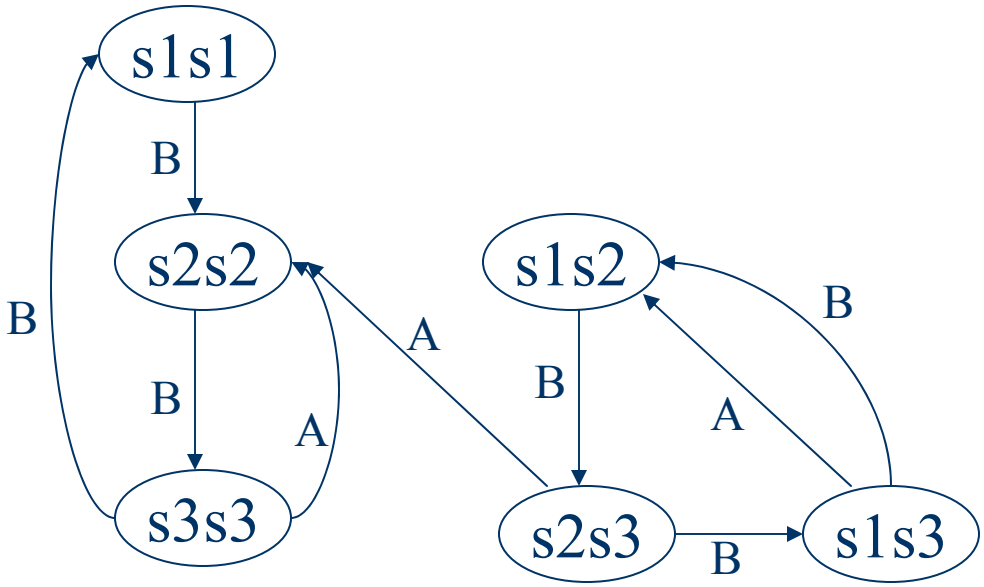
Problem #1: Determining the final state after the test

- Existence: Knowing whether one exists
  - Homing sequence - All deterministic finite state machines has a homing sequence. Process of elimination. Start current state uncertainty with one block with all the states. Let each sequence partition the current state into two blocks, each of which correspond with different output. Repeat ‘til one state is reached.
  - Synchronizing sequence – Draw an auxiliary graph GxG, a node for every unordered pair. There is an edge from (si, sj) to (sk, sm) if input a takes si to sk AND sj to sm. Check if there is a state (si, si) that all states can reached.

Problem #1: Determining the final state after the test

Auxiliary graph:





{s1s2s3}, take (s2s3) starting, give input A.

Observe the $\delta(S, A) = \{s1s2\}$.

Take (s1s2), give input BA.

Synchronizing sequence is ABA.

Problem #1: Determining the final state after the test

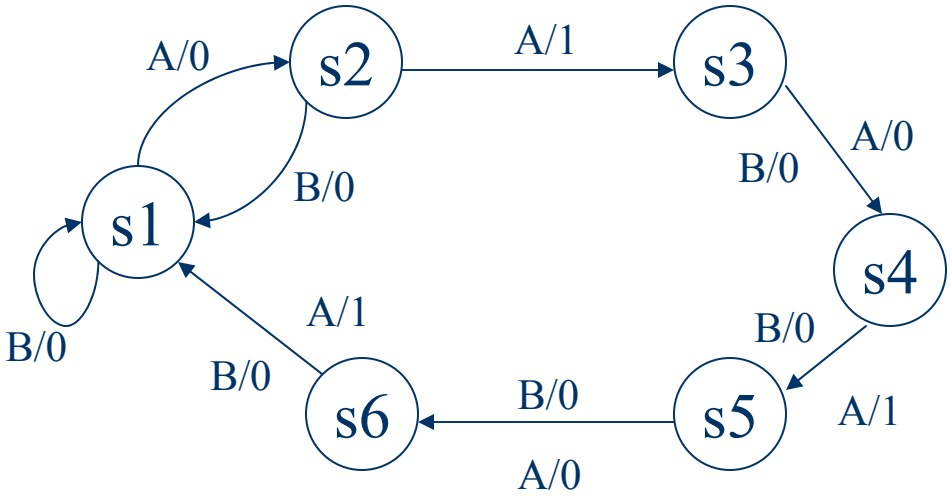
- Building and finding one:
  - Homing sequence: Construct a homing sequence from the successor tree by finding the node of least depth, the current state uncertainty consists only of singletons (one element only). Shortest homing sequence is an NP complete problem.
  - Synchronizing sequence: After auxiliary graph is constructed, use BFS to check reachability condition. If condition is true, take two states where si != sj, find shortest path from a node to the final node (sr,sr). Consider that input X_1. Now $\delta(S, X_1)$ = set of S – si (or sj). Take result from transition, and repeat, record the input, 'til there is only the final state left. The sequence is concatenation of $X_1X_2X_3$….

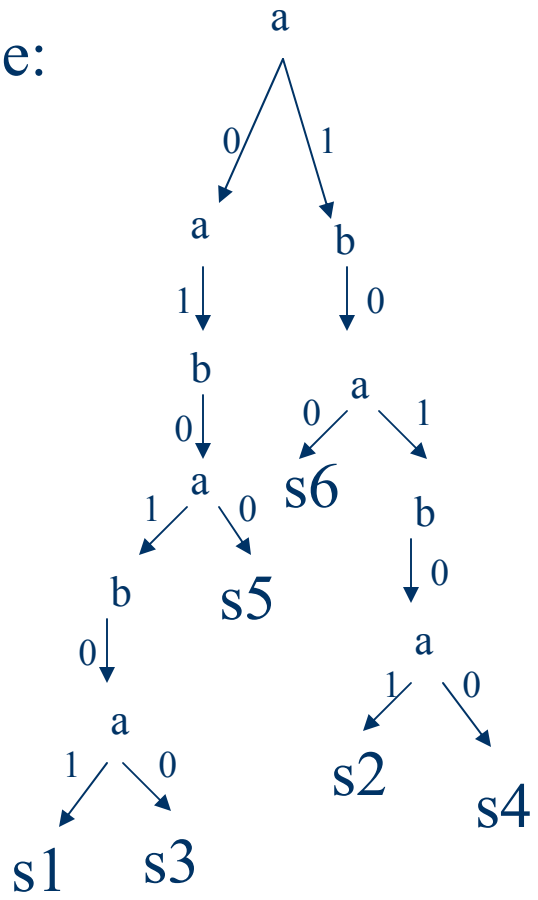
Problem #2: Identify the unknown initial state

- Assume we know the complete state diagram of a machine M, but we do not know its initial state. The problem is to find the initial state. This is not always possible.
- Solution: Distinguishing sequence
  - An input sequence that solves this problem. Two types, preset and adaptive distinguishing sequence. Some machine has no distingushing sequence, some has only adaptive ones, others has both adaptive and preset.
- Existence, building and finding one:
  - Preset distinguishing sequence: Examine successor tree, annotate the node of the tree with its initial state uncertainty instead. A distinguishing sequence only exist if one sequence at the end has all blocks of its initial state uncertainty being singletons.

Problem #2: Identify the unknown initial state

Ex. An adaptive distinguishing sequence:



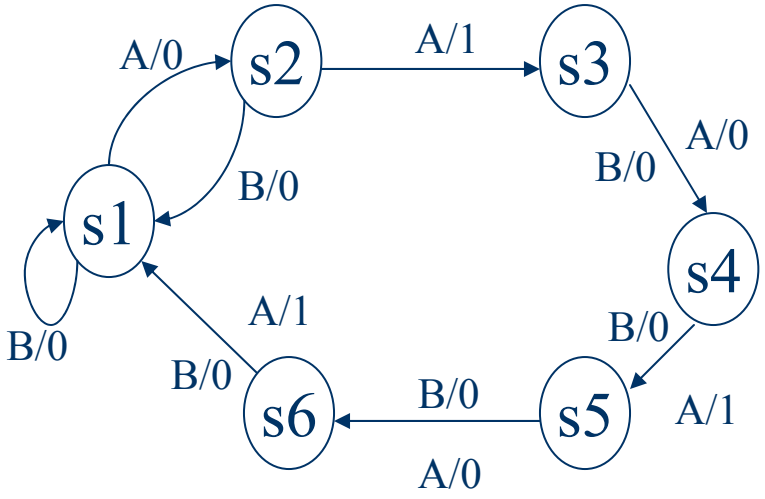


Problem #2: Identify the unknown initial state

Existence: Adaptive distinguishing sequence.

- Check if the input can generate different output. If so, partion the block
- If the input can cause a state to move to a different state that's not in the original block, partition.
- Recursive check on the blocks.

– Ex. A on set {s1s2s3s4s5s6} = {{s1s3s5}{s2s4s6}}
B->1st block, {{s1}{s3s5}{s2s4s6}}
A->3nd block, {{s1}{s3s5}{s2s4}{s6}}
A->2nd block, {{s1}{s3}{s5}{s2s4}{s6}}
B->4th block, {{s1}{s3}{s5}{s2}{s4}{s6}}



Problem #2: Identify the unknown initial state

- Building and finding an adaptive distinguishing sequence:
  - Use a two part algorithm. First part generate a splitting tree, that reflect the sequence of blocks splitting, and second part takes the splitting tree and generate the adaptive distinguishing sequence.
  - First part: A splitting tree is a tree where every node is a set of state. The root contain the whole set of states. Each leaf is a singleton, each node (nonleaf) is the union of its childrens. Also, every node is associated with an input sequence, every edge is associated with an output symbol. For every node/block, a valid input does the following:
    - Case I) States in the block produce different output
    - Case II) All state produce same output, but map to more than one block from original.
    - Case III) All state produce same output and mapped into same block.
- In essence, same as the operation we did to prove its existence.

Problem #2: Identify the unknown initial state

Ex. A on set {s1s2s3s4s5s6} =
{{s1s3s5}{s2s4s6}}

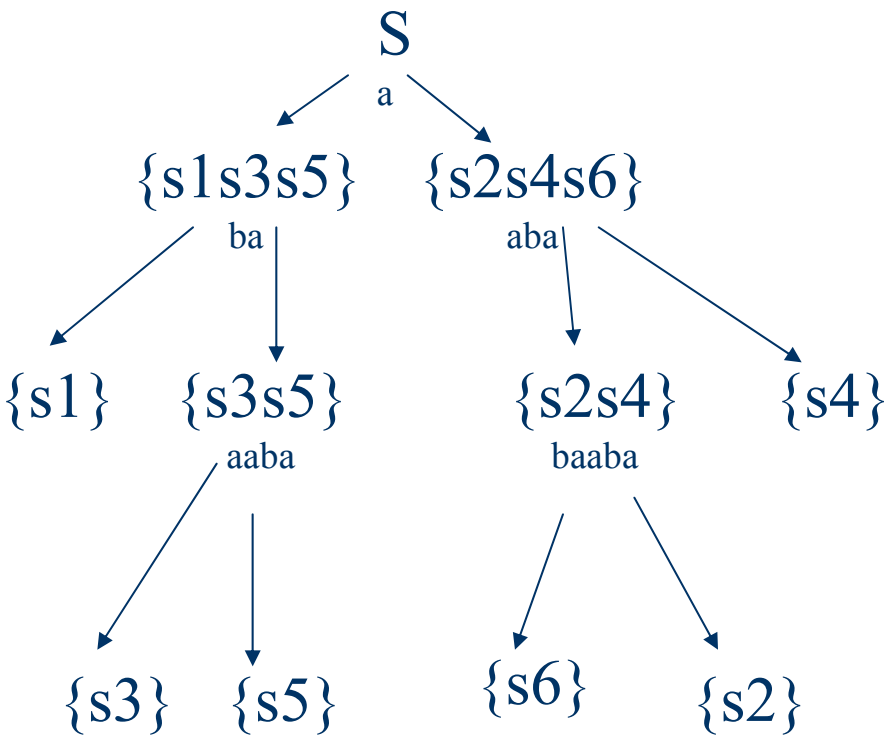
B->1st block, {{s1}{s3s5}{s2s4s6}}

A->3nd block,{{s1}{s3s5}{s2s4}{s6}}

A->2nd block, {{s1}{s3}{s5}{s2s4}{s6}}

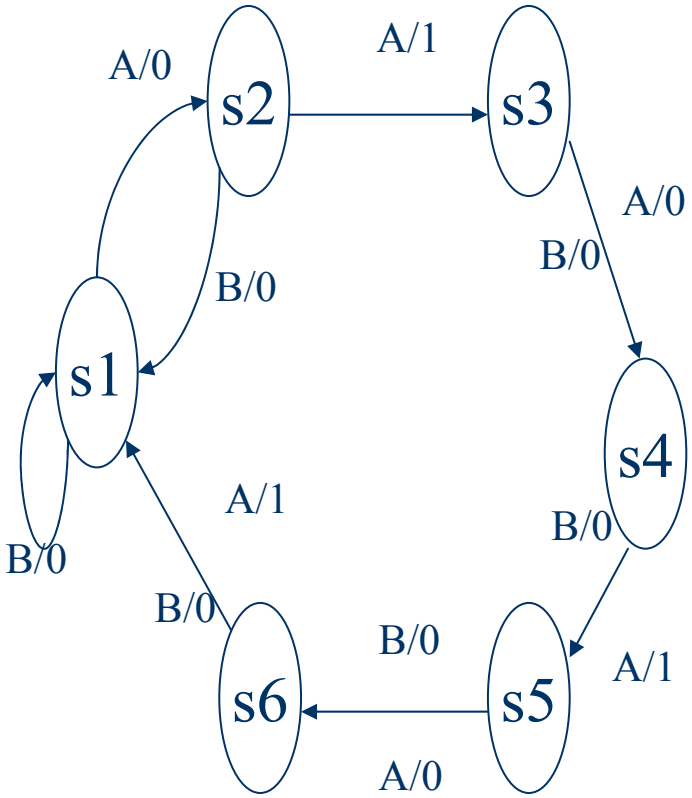
B->4th block, {{s1}{s3}{s5}{s2}{s4}{s6}}

Note that the sequence attached to each of the blocks is cumulative in how the tree is constructed.



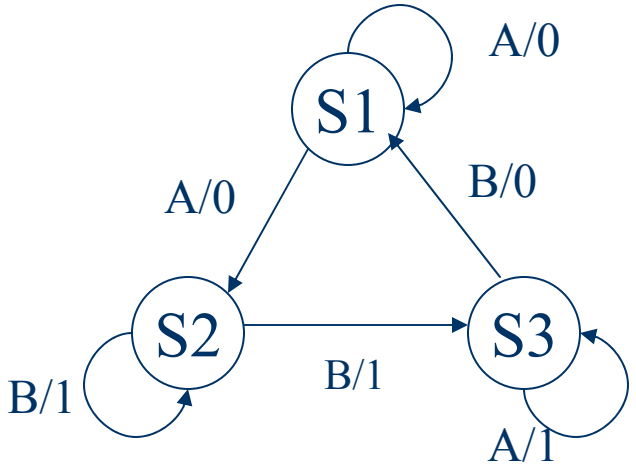
Problem #2: Identify the unknown initial state

- Part II: Have the set initial and current states be available so you can match each other. Find the lowest node who contains the current set, apply the input sequence associate with the set, observe output. Repeat.
- Ex. Input a. Output is 1, we know initial uncertainty is {s1s3s5}, current uncertainty is {s2s4s6}. Find node with {s2s4s6}, who has a sequence of aba. Input aba.
- Output is one. We know the only possible current state from previous state is {s1s5}. Look up {s1s5}, sequence of ba. Input ba. Initial state located.



Problem #3: State verification

- Just to clarify the question, we want to verify that a given machine M with a known state diagram is/was in a particular state.
- Solution UIO sequence
  - UIO sequence of a state S is an input sequence such that the machine would produce a unique output sequence if and only if the machine were in state S.
- Example:
  - B is UIO for s1
  - s2 has no UIO
  - A is UIO for s3

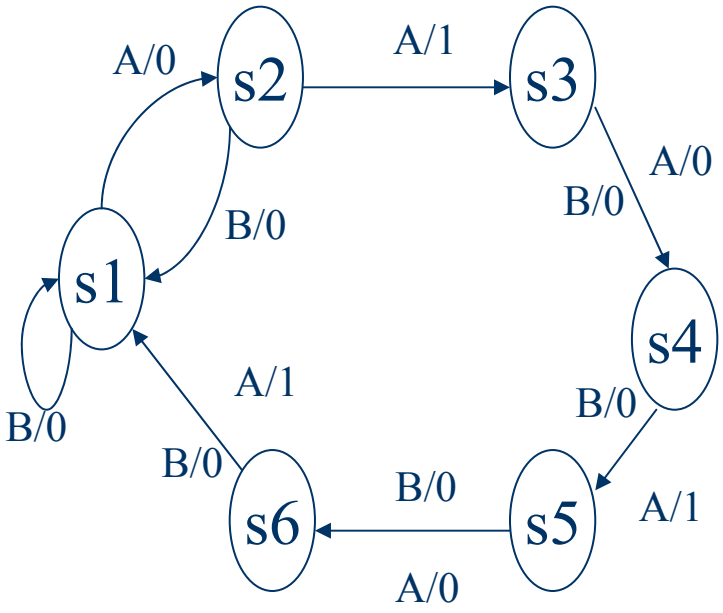


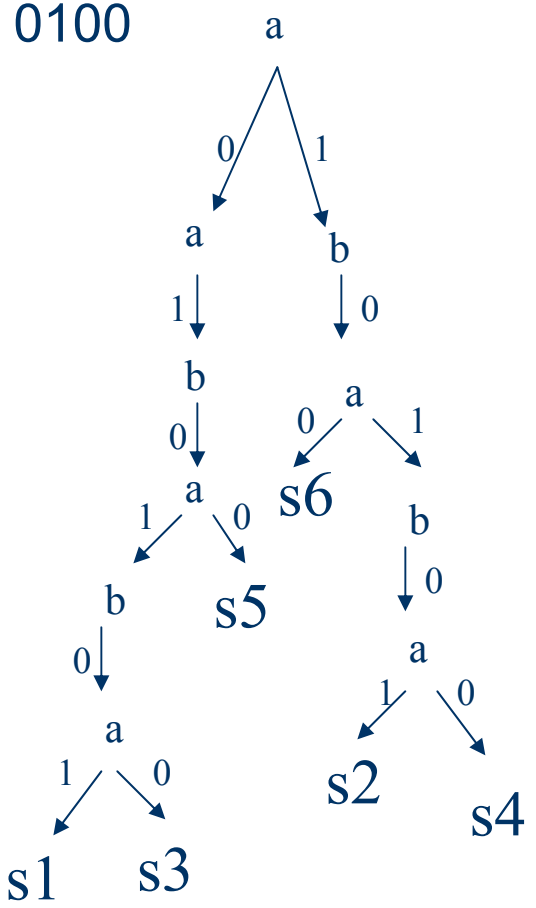
Problem #3: State verification

- Existence: There is no efficient algorithm for finding UIO sequence.
- However, if an FSM has distinguish sequence, that means every state has an UIO sequence. Its UIO sequence is the input from the root of the deicison tree to a leaf, which correspond to the state.

Problem #3: State verification

Ex. S6: aba 100, S2: ababa,10101 s5: aaba 0100





Problem #4: Conformance Testing

- Also known as fault detection. We have complete specification of a machine A, including its state diagram. We are given a machine B that is a “black box”, and we can only observe its I/O behavior. Design a test to determine whether machine B is a proper implementation of A.
- Couple of assumptions we are making for test to be possible:
  - A is strongly connected (you can get from one state in A to another state thru some sequence)
  - A is a reduce FSM
  - Implementation of B does not change during the experiment and has same input alphabet as A
  - Machine B has no more state than A

Problem #4: Conformance Testing

- Now we can start use all the tools we learned:
  - First, apply a homing sequence that's suppose to bring B to a known state s1, the initial state of the test.
  - Run *checking experiment*, verify B is equivalent to A. Check output. If B isn't isomorphic to A, then homing sequence may or may not have bring B to s1. Either way the checking experiment will detect faults.
- Solution: A checking sequence is an input sequence such that it distinguish A from all other machine with same number of states, starting from initial state s1. All machine that are not isomorphic to A, on input x produce a different output.
- All checking experiment has same structure
  - For every transition of A, from s_i to s_j on input X, apply an input sequence to get machine to state s_i, apply input A, verify end state is s_j.

Problem #4: Conformance Testing

- There's a lot that goes into comformance testing, but here's a few basic concepts:
  - *Separating family of sequence* is a collection of sequence such that for every pair of states si, sj, there is an input that separates them.
  - *Status message* tells us the current state of the machine.
  - *Reset capability* is taking an input symbol takes the machine from any state back to initial state s1.
  - *Transfer sequence* is a sequence take the machine from one state to another, such a sequence always exist since the machine is strongly connected.
  - *Identifying sequence* is a sequence that identify a state in the middle of execution.

Conclusion

Problem	Solution	Length
Final state	Homing	$n(n-1)/2$
Final state	Synchronizing	$n(n^2-1)/6$
State ID	Distinguish (preset)	Exponential
State ID	Distinguish (adaptive)	$n(n-1)/2$
State verification	UIO	Exponential
Conformance	Checking	$pn^3+\min(p,n)n^4\log n$