Testing, Optimization, and Games

Mihalis Yannakakis
Columbia University

The Software Reliability Problem

Systems are becoming larger, more complex,distributed,…

$\Rightarrow$ harder to create, get them right, test them …

- Large part of the cost of software development goes to testing

Problem: Improve cost, time, reliability

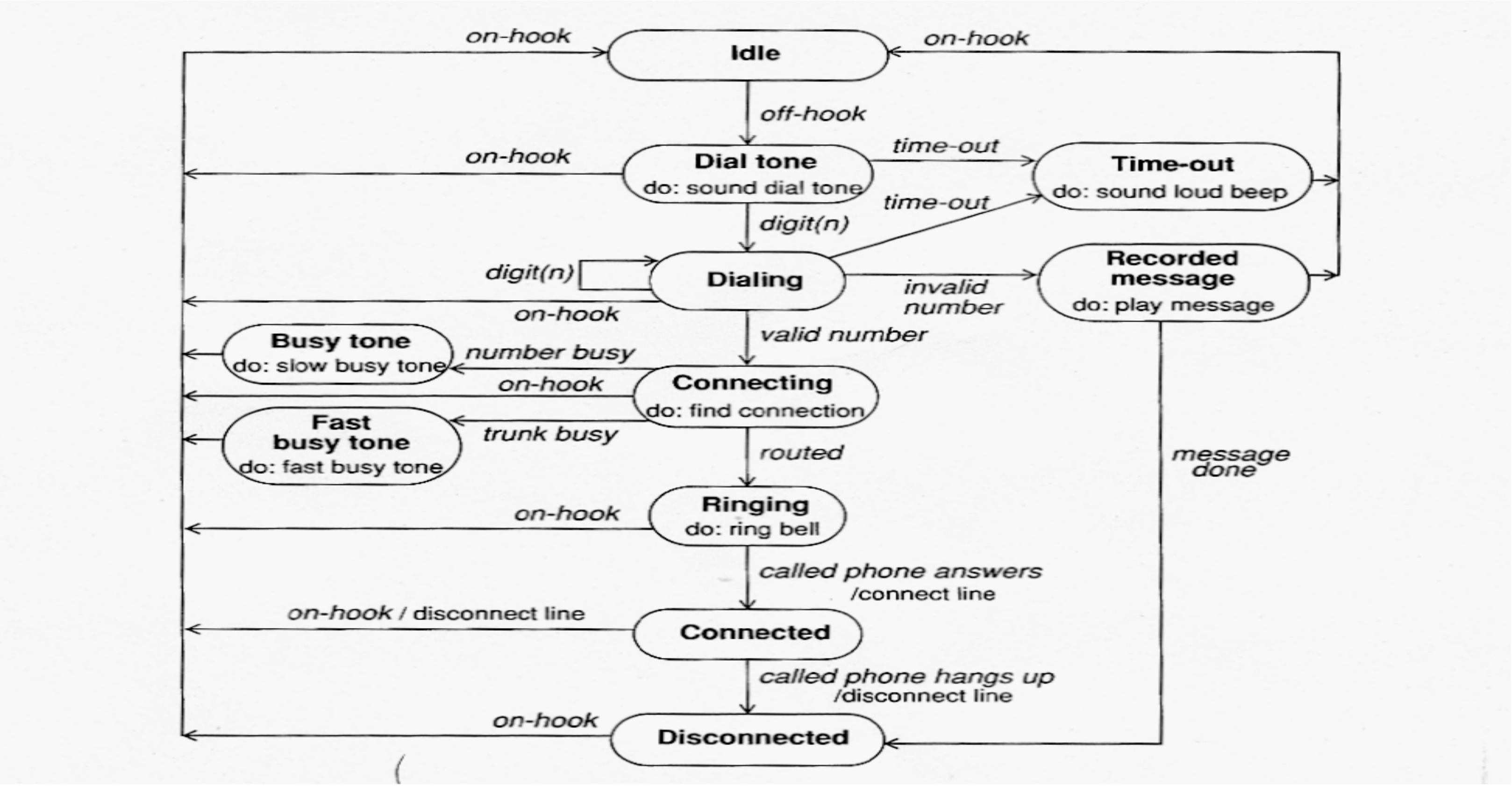
Focus: Behavior/Control of Systems

Reactive/Event-driven Systems

- Switching Software
- Communication Protocols
- Controllers
- ….

Model: State Machines of various types

Finite State Machine for Phone

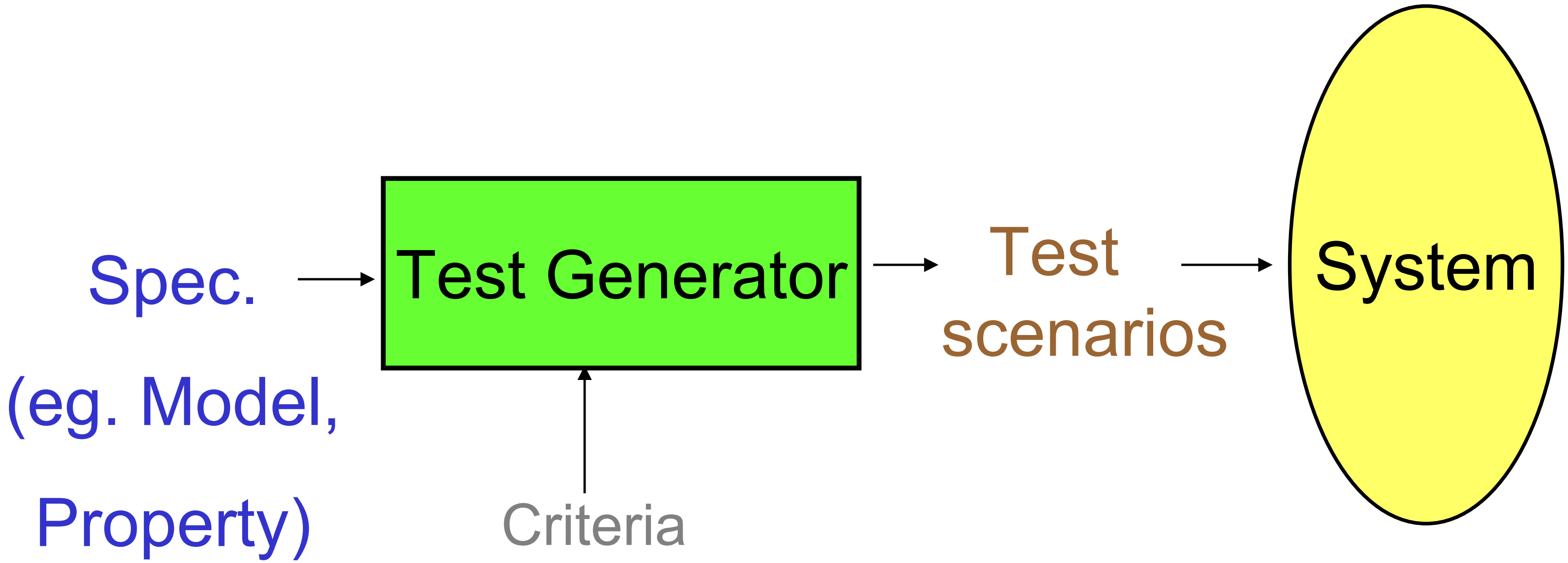


States: Idle, Dial tone, ….

Inputs: off-hook, on-hook, digit, …

Outputs: sound dial tone, loud beep, play message,….

Testing



Does the System satisfy the specification?

(conform to the model ? satisfy the property?)

Different Views of Testing

- Testing as an Optimization problem

  Optimize the use of testing resources to achieve maximum fault coverage

- Testing as a Game

  Tester vs. System

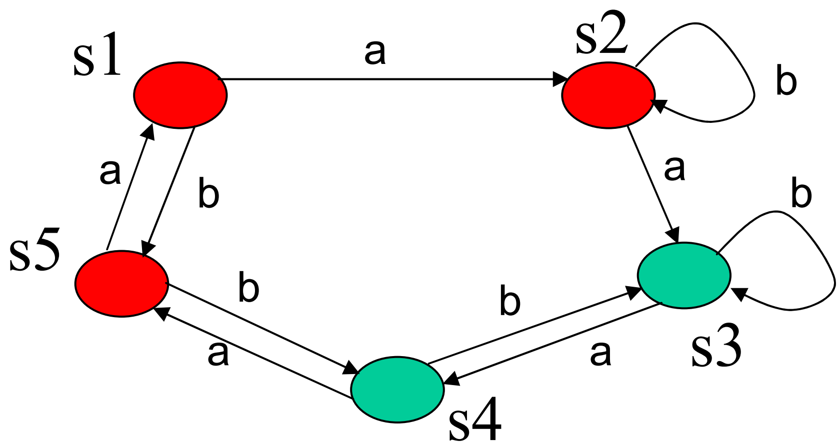
  Who wins? Best strategy?

- Testing as a learning problem

Outline

- Testing framework, issues
- Conformance Testing
  - Deterministic FSM’s
  - Nondeterministic FSM’s
- Testing Properties
- Optimum Coverage problems
  - FSM’s, graph models
  - Extended FSM’s
  - Hierarchical FSM’s

Finite State Machine



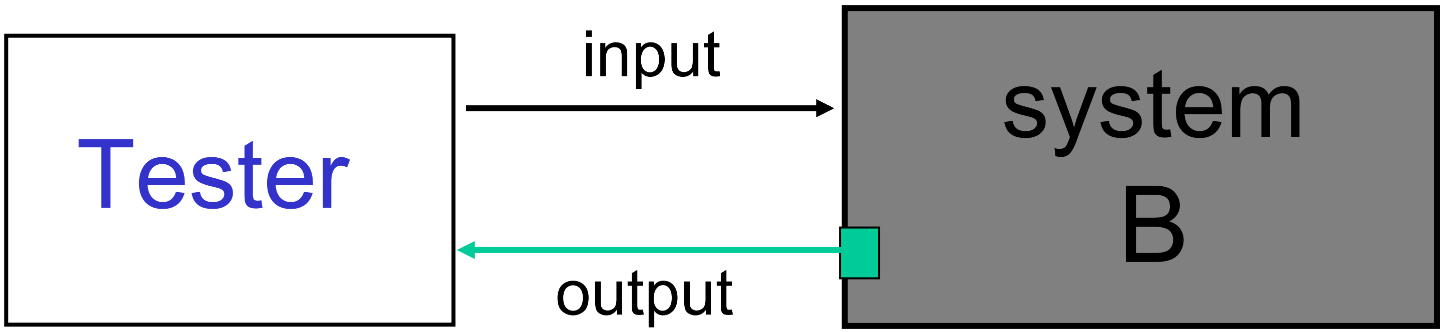
Moore machine

- States: s1, …., s5
- Inputs: a, b
- Outputs: red, green - function of the state
- Transitions: for every state and input

Deterministic FSM: one transition for every state and input

Mealy machine: variant where outputs are produced on transitions instead of states; theory is similar

Test



Problem: Given some a priori information about B, compute a desired function of B

Preset Test: input sequence selected ahead of time

Adaptive Test: inputs selected online adaptively, i.e. can depend on previous outputs

Testing as a Game

- Game:

  1. A priori information ("testing hypothesis"): Set U of possible B's
  2. Desired information: function f of B

- Players:

  - Tester: selects inputs, gives verdict at end
  - System: Selects B in U, and moves of B in each step (if B not deterministic)

- Tester wins if verdict=f(B)

- Game with incomplete information

Questions

- Can the Tester always win?

  i.e. ∃ strategy (test) that arrives at correct result?

Questions

- Can the Tester always win?

  i.e. $\exists$ strategy (test) that arrives at correct result?

- How fast can we determine if the Tester has a winning strategy?

Questions

- Can the Tester always win?

  i.e. ∃ strategy (test) that arrives at correct result?

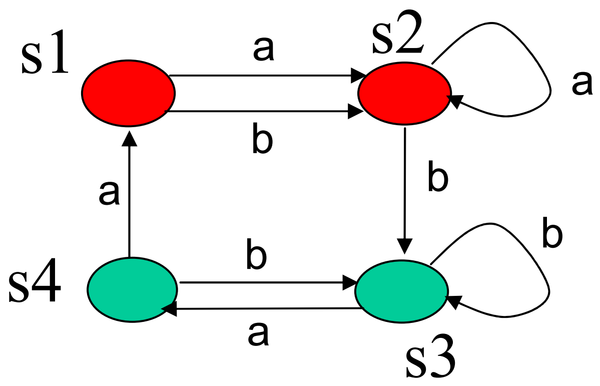
- How fast can we determine if the Tester has a winning strategy?

- What is the *testing complexity* = length of the test (winning strategy)

- and the *computational complexity* = time to compute a winning strategy?

Example: Adaptive Distinguishing “Sequence”

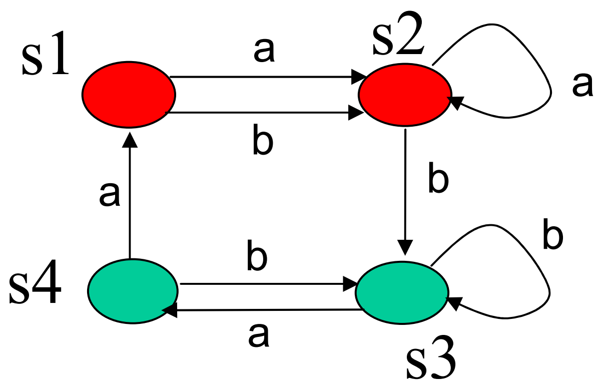
Given:



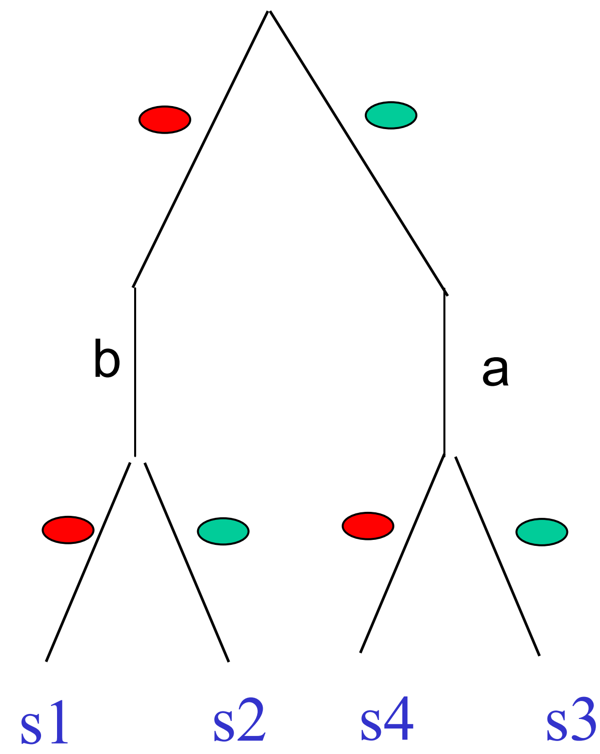
State diagram of B = a deterministic FSM

Goal: Determine the initial state of B

Example: Adaptive Distinguishing “Sequence”



FSM

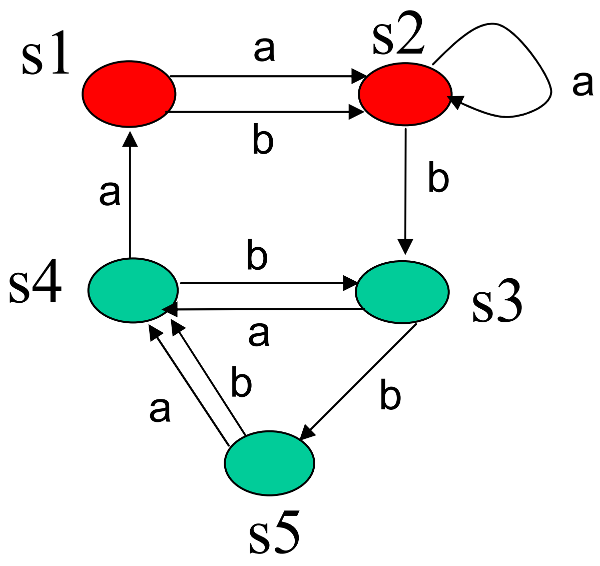


adaptive distinguishing “sequence”

= winning testing strategy

Questions

- Can the Tester always win?
  - No (not even if FSM is reduced, i.e. has no equivalent states)



Questions

- Can the Tester always win?
  - No (not even if FSM is reduced, i.e. has no equivalent states)
- How fast can we determine if the Tester has a winning strategy?
  - O(dnlogn), n=#states, d=#inputs
  - For Preset test: PSPACE-complete

Questions

- Can the Tester always win?
  - No (not even if FSM is reduced, i.e. has no equivalent states)
- How fast can we determine if the Tester has a winning strategy?
  - O(dnlogn), n=#states, d=#inputs
- What is the *testing complexity* = length of the test (winning strategy)
  - $O(n^2)$
- and the *computational complexity* = time to compute a winning strategy?
  - $O(dn^2)$
- Preset: Exponential

[Lee-Yannakakis]

Unknown state diagram of black box B

- Machine Identification Problem:
- Given:
- B is a reduced (minimized) deterministic FSM

  (tests cannot tell the difference between equivalent machines)

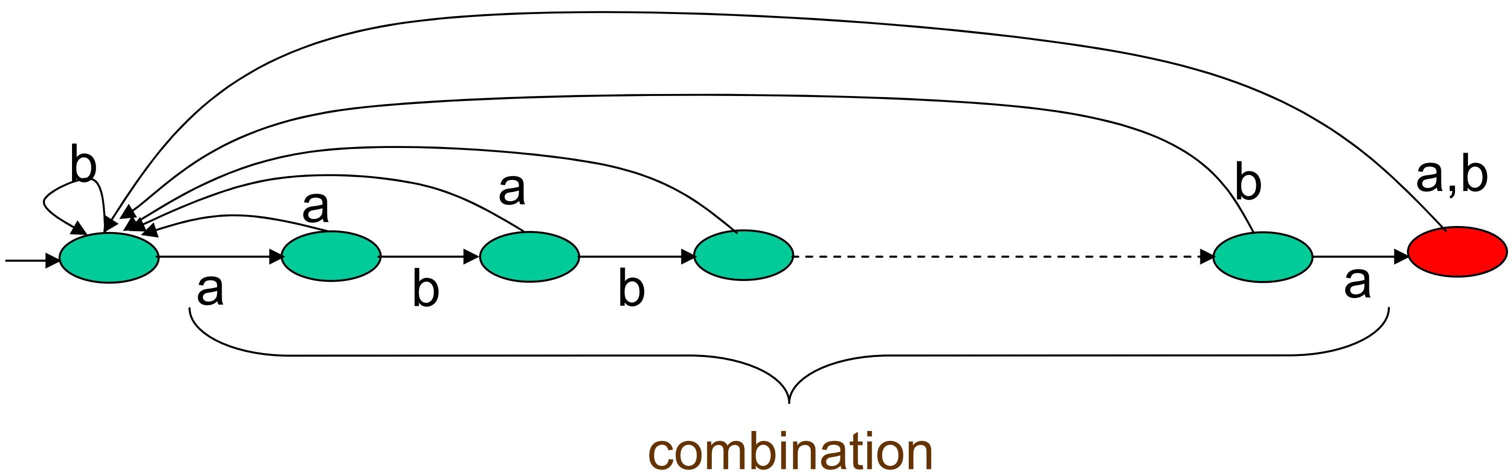
  - and strongly connected

  (i.e. any state can reach any other state)
- bound on # states of B

Goal: Identify machine B

Machine Identification is hard

- Suppose that we know B has *n* states and looks like this combination lock machine



Must try all possible combinations: d^{n-1}

$d =$ # inputs, $n =$ # states

[Moore]

Conformance Testing

- Given: specification FSM A

- Goal: check that B conforms to (behaves like) A

  (i.e. B$\equiv$A for deterministic FSMs)

- Long History since 50's [Moore, Hennie,…]

Conformance Testing - Deterministic FSM

Assumptions

- Specification machine A is reduced (minimized)

  (tests cannot tell the difference between equivalent states)

  and strongly connected

  (i.e. any state can reach any other state)

- Bound on #states of B

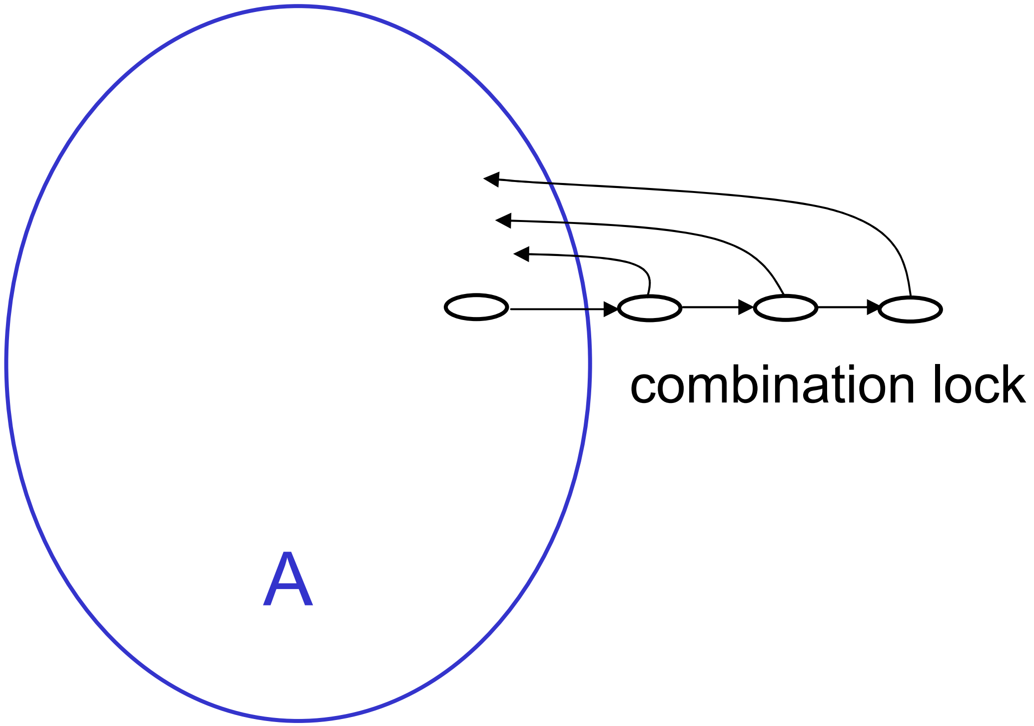
- Checking sequence: If implementation machine B has no more states than A: detect arbitrary combinations of *output*, and *next-state* faults

  - effect of extra states orthogonal

Effect of extra states

Extra factor of d^k, where k =#extra states, d=# inputs

B :



Questions

- Can the Tester always win?
  1. Can test that B has the same state diagram as A
  2. But in general may not be able to verify the initial state (if no reset) even if we know state diagram of B

- Can perform a test such that if B passes it, then can conclude that B≡A and B is at an equivalent state *at the end* of the test

Easy cases

- Spec FSM A is fully observable:

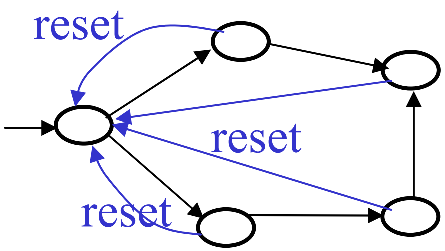
  every state has a distinct output $\Rightarrow$ suffices to traverse all the transitions

- Spec FSM A has a distinguishing sequence:

  $\Rightarrow$ checking sequence of length $O(dn^3)$

[Hennie,LY]

Machines with Reliable Reset



- There is a special input symbol "*reset*" which takes every state back to the initial state
- *Reliable*: works properly in the implementation FSM B

- Then checking sequence of length $O(dn^3)$
- Matching lower bound

[Vasilevski- Chow]

General machines

- Randomized polynomial time algorithm which, given a specification machine A constructs with high probability a checking sequence for A of length $O(dn^4 \log n)$ [LY]

- For "almost all" specs A, length O($d \cdot n \cdot$polylog$n$)

- Deterministic algorithm?

Sketch of (simplified) Test

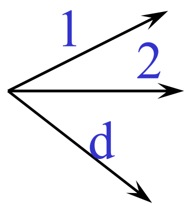
- Pick a set W of "separating" input sequences such that every pair of states of the spec FSM A is distinguished by one of these sequences
  - There is always such a set of at most n sequences of length at most n

Repeat the following "enough" times

- Choose at random a transition (state s, input *a*)
- Apply an input sequence that takes A from the current state to state s
- Decide at random whether to check the state of B or check the transition
  - In the first case, apply a random separating sequence from W
  - In the second case, apply input *a* followed by a random separating sequence from W

A universal traversal problem

Directed graphs with n nodes, outdegree d



- Blocking sequence over {1,...,d}:
  For every graph and starting node,
  path traverses all edges out of at least one node.

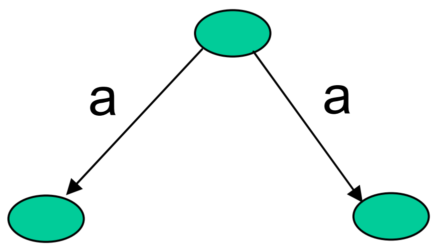
- Random sequences of polynomial length blocking

- Deterministic polynomial construction?

Then deterministic construction of checking sequence for all spec FSM's

Nondeterministic FSM

Many possible transitions for same input and state

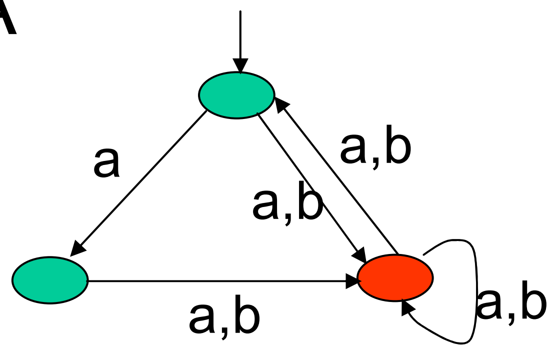


- Nondeterminism in spec A: multiple acceptable choices
- Nondeterminism in system B: some transitions are not under tester's control
  - abstraction, other entities, concurrency, ..

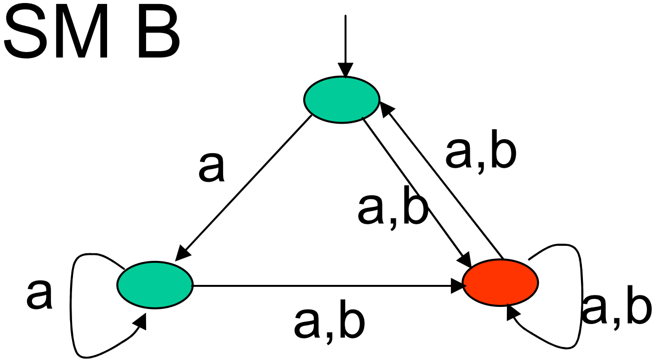
FSM B conforms to FSM spec A if every response to any input sequence could have been produced by A

Example

Spec A



FSM B



- B does not conform to A:

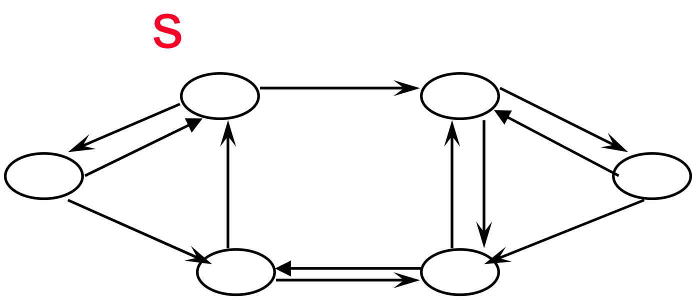
  On input aa , B *may* output , but not A

- B may also output  or  or  which are consistent with A

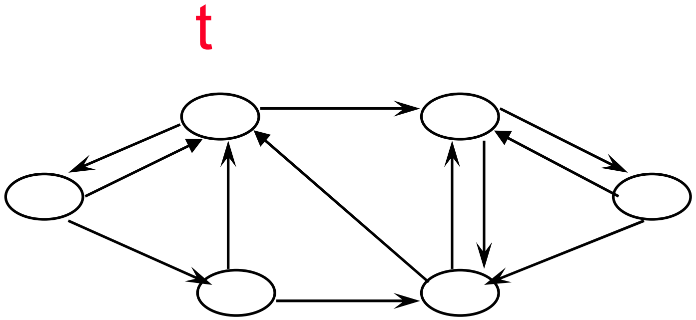
Distinguishing Between Machines

Spec A

(correct FSM)



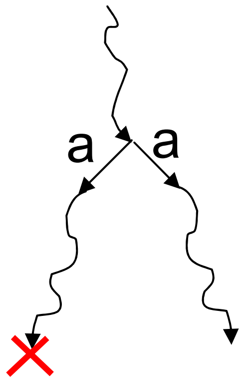
Possible faulty FSM B



Two-player game

- Tester chooses inputs
- System player chooses what's in the black box and *how to resolve the nondeterminism*

- Should we view the system player as trying to
  - Help the tester?
  - Oppose the tester?
  - Indifferent (random)?



Opposing System Player

- Tester has winning strategy ⇔ can find a fault (if present) no matter how hard the system tries to hide it

⇔ Games with incomplete information against a malicious adversary

- Game graph of positions, controlled by the two players
- Player 1 gets only partial information about current position
- Goal of Player 1: reach a winning position

Who wins?

- preset test: PSPACE-complete
- adaptive test: EXPTIME-complete
- Polynomial time for NFSM that are input-output deterministic (observable)

[Reif; Alur, Courcoubetis, Y]

Indifferent System player: Random moves

If the system has *reliable reset,* then easy: can test with probability $\rightarrow 1$

B does not conform to A $\Rightarrow$ for some input sequence α it can produce (for some nondeterministic path) an output sequence that can't be produced by A

Test: Apply repeatedly reset α , reset α, ….

Indifferent System player: Random moves

In general, Game with incomplete information against "Nature" (a Random adversary)

- Partially observable Markov Decision Process
  - maximize probability of reaching goal
  - can we reach goal a.s.?

Can the Tester win with probability 1 (in the limit)?

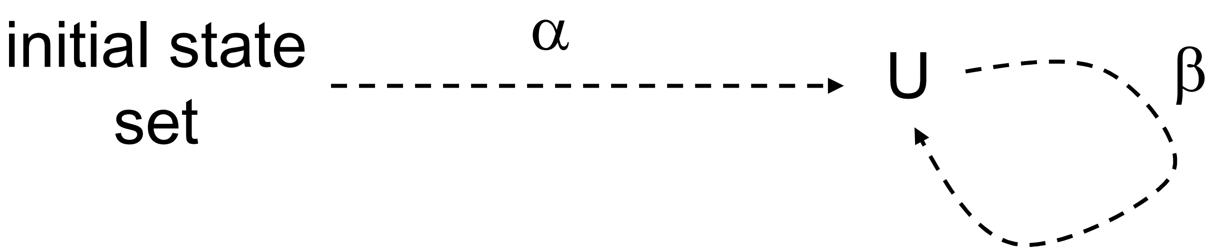
- Complexities similar to adversarial game – algorithms different

[ACY]

Blindfold (Preset) Game

- Given FSM M of the game, construct deterministic FSM D
- There is an a.s. winning strategy iff

  there is a state U of D and input words α, β s.t.

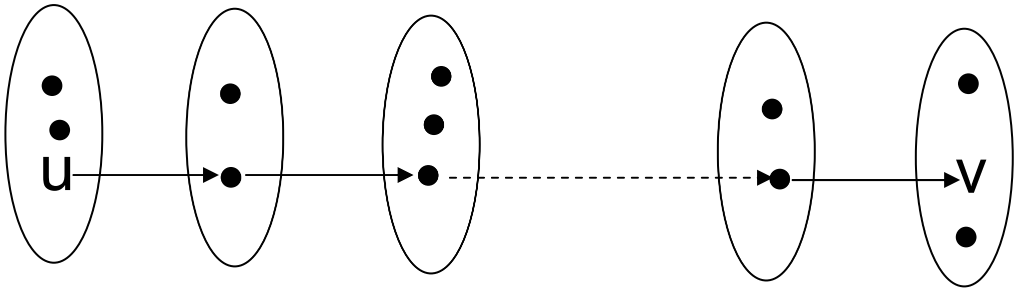


and for all u in U, M has a path u $\xrightarrow{\beta}$ winning position

- Winning strategy: $\alpha\beta\beta\beta\beta\ldots$

Adaptive Strategies

- Regard the game FSM M as a Mealy machine (transfer state color output to incoming edges)
- Construct graph G that is deterministic wrt input-output pairs, by subset construction
- G will be iteratively pruned until it gets stabilized. At any point, say that state u of M reaches state v in G iff the current G has a path



Adaptive Strategies ctd.

- Repeat until no change

  - Choose a node U of G with a state u in U s.t. no winning state v is reachable from u in G; if no such u exists then terminate

  - If there is an edge in G from a node V into U with input label a, then delete all edges from V with input label a

  - Delete U

- There is an a.s. winning strategy iff the final graph G contains the initial state set

Testing Properties

Testing Properties

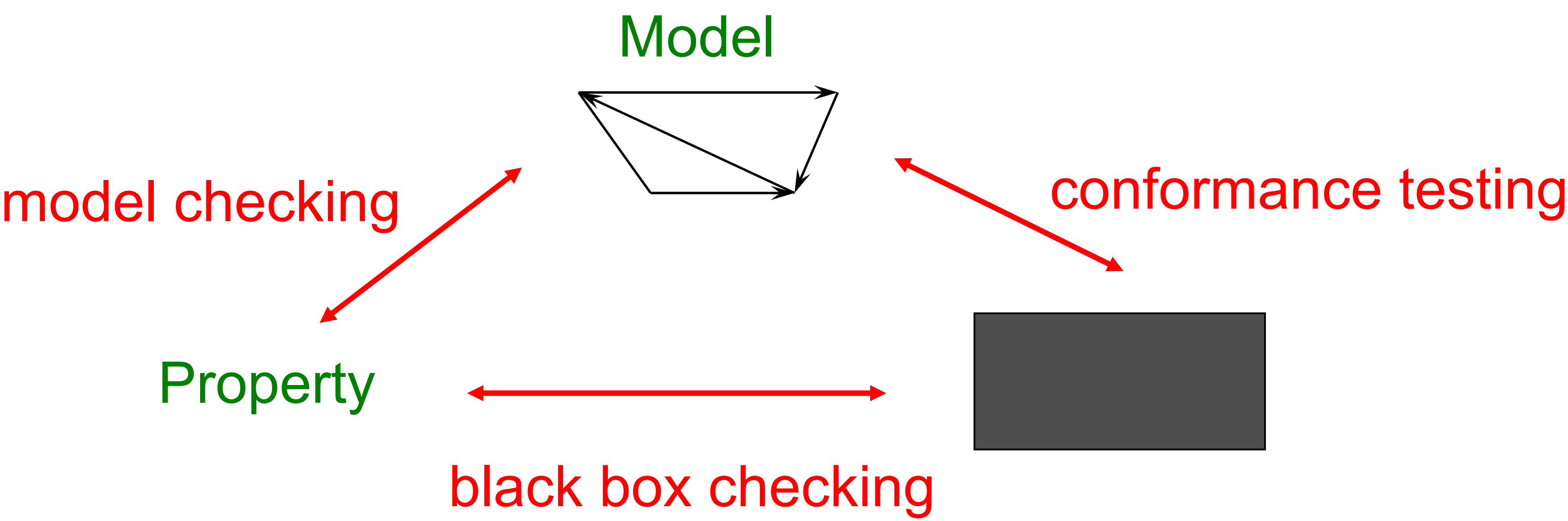
- Given a required property of executions
  - e.g., if off-hook then dial-tone; no deadlock …
  - between any two green states always a red state
- and a black box B (the system)

Test that B satisfies the property

Testing Properties

- Given a required property of executions
  - e.g., if off-hook then dial-tone; no deadlock …
  - between any two green states always a red state
- and a black box B (the system)

Test that B satisfies the property



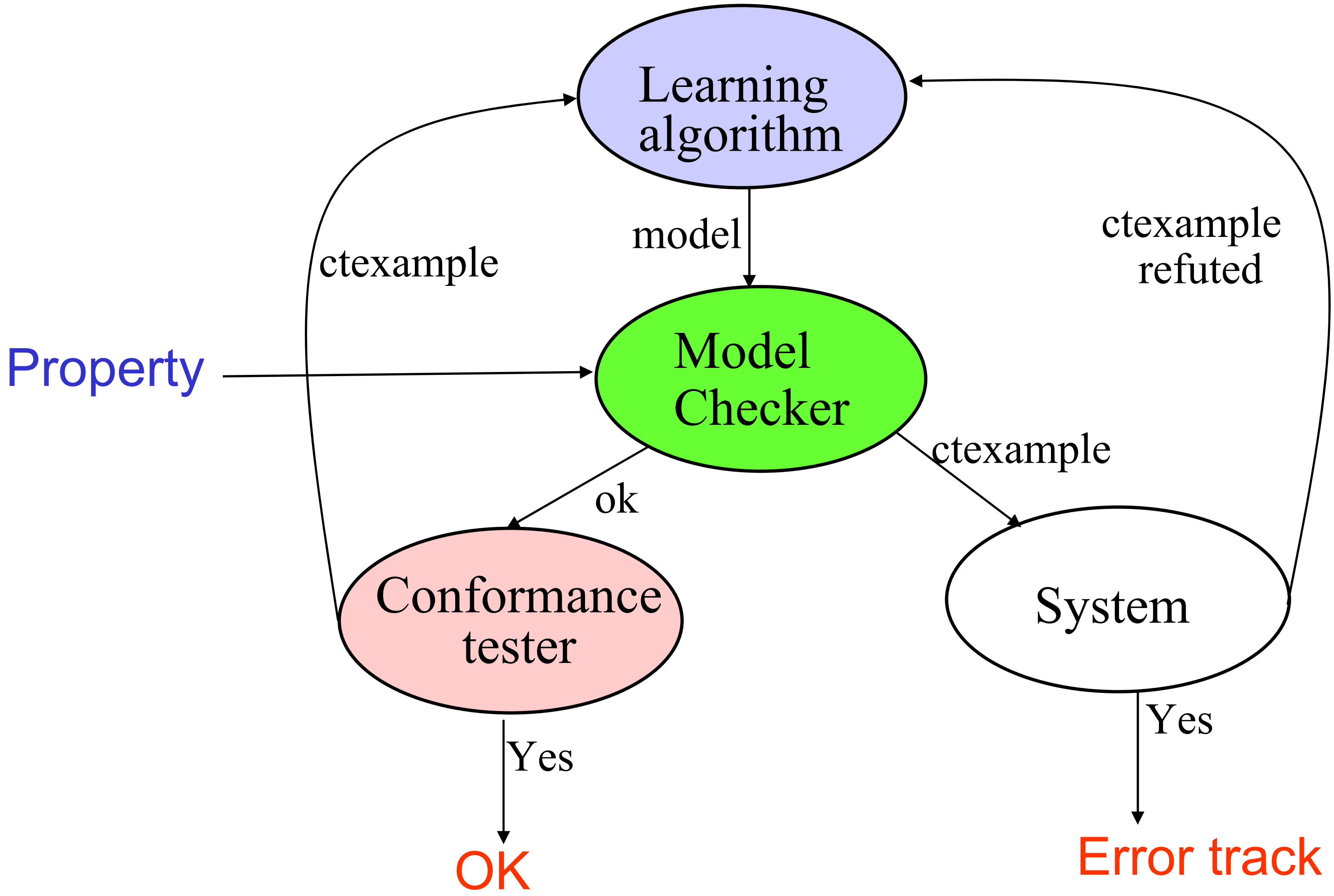
[Peled, Vardi, Yannakakis]

Learning FSM with a teacher

- Algorithm to identify a deterministic FSM using
  - “membership queries” (tests) on the black box
  - “equivalence queries” to the teacher

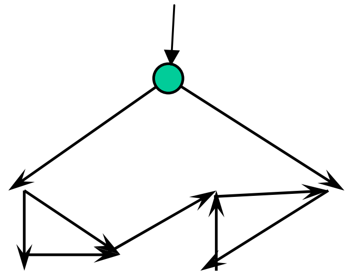
- FSM with reset: polynomial algorithm [Angluin]
- General FSM: randomized polynomial algorithm [Rivest –Shapire]

Black Box Checking



Optimization

Optimal Coverage Problems



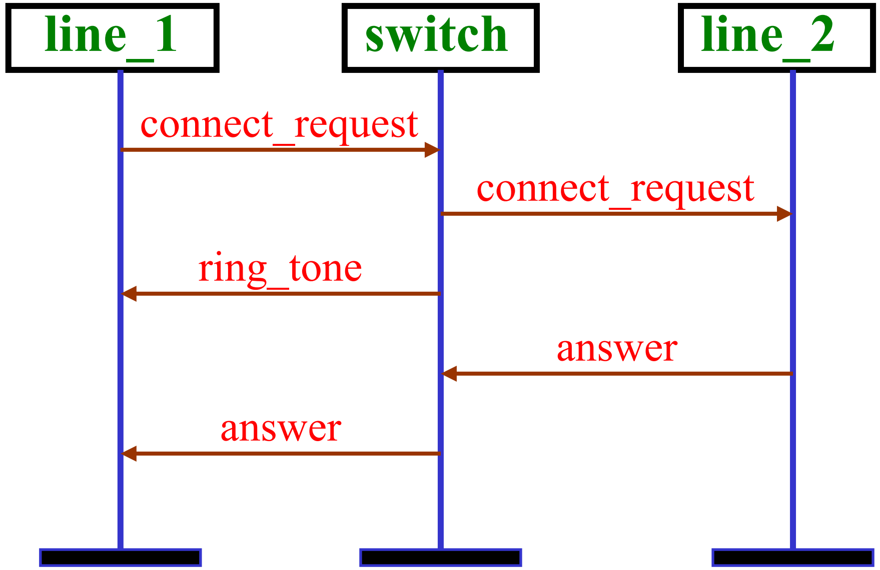
- Find a minimum number of short test sequences (paths) starting at initial state that cover all transitions, states ......

- Applies to FSM models and other graph models
- Use Case (MSC) Graphs: scenario based models

  uBET - Lucent Behavior Engineering Toolset

Scenario-based Models

1. Message Sequence Charts

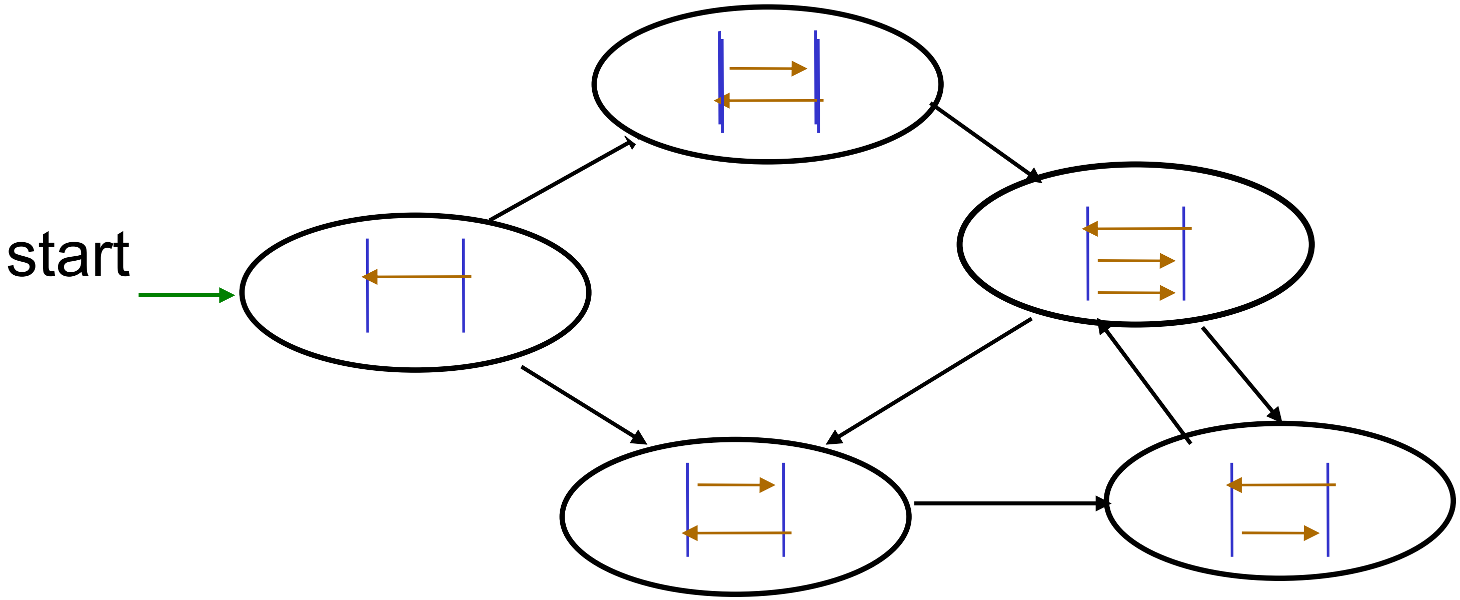


Partial order of events

Scenario-based Models ctd.

2. MSC Graphs

Graphs whose nodes are MSC's on the same processes

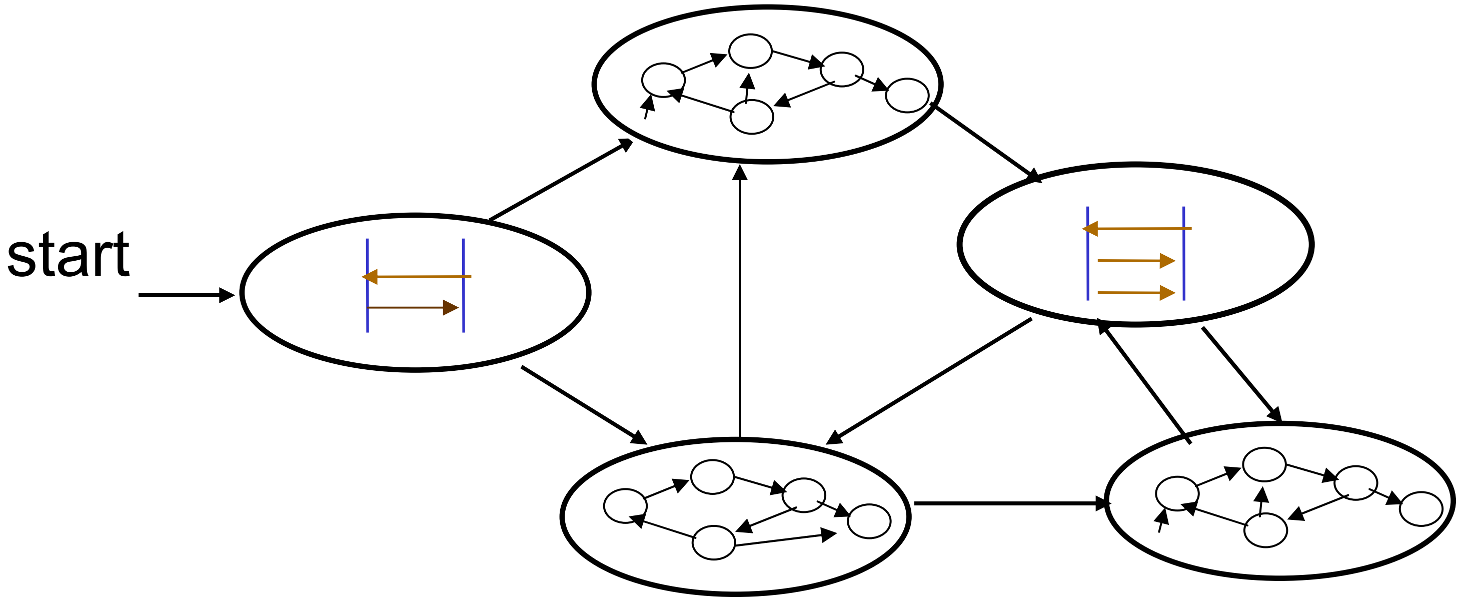


Path through the graph → combined message sequence chart

Scenario-based Models ctd.

3. Hierarchical MSCs

Graphs whose nodes are MSC's or nested HMSC's



Path through the nested graphs → message sequence chart

uBET - Lucent Behavior Engineering Toolset

Graph Coverage

- Transition Coverage

  Can minimize in PTIME

  (1) the number of paths,

  (2) their total length, subject to (1)

  (or any linear combination of 1 and 2)

  - Network flows, Chinese Postman Problem

- State Coverage

  Can minimize the number of paths

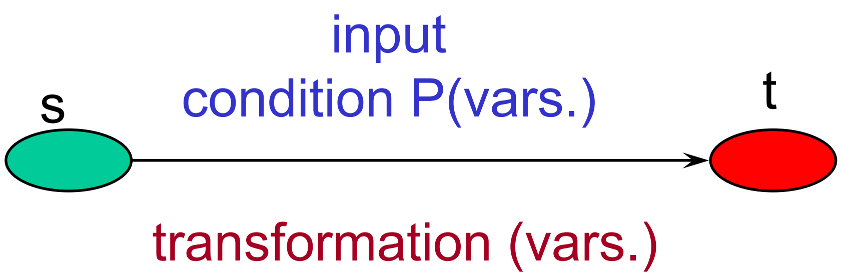
  but not the length

  - Asymmetric Traveling Salesman Problem

Extended Finite State Machine

FSM + variables

- States
- Variables
  (Boolean, arithmetic, …)
- Transitions

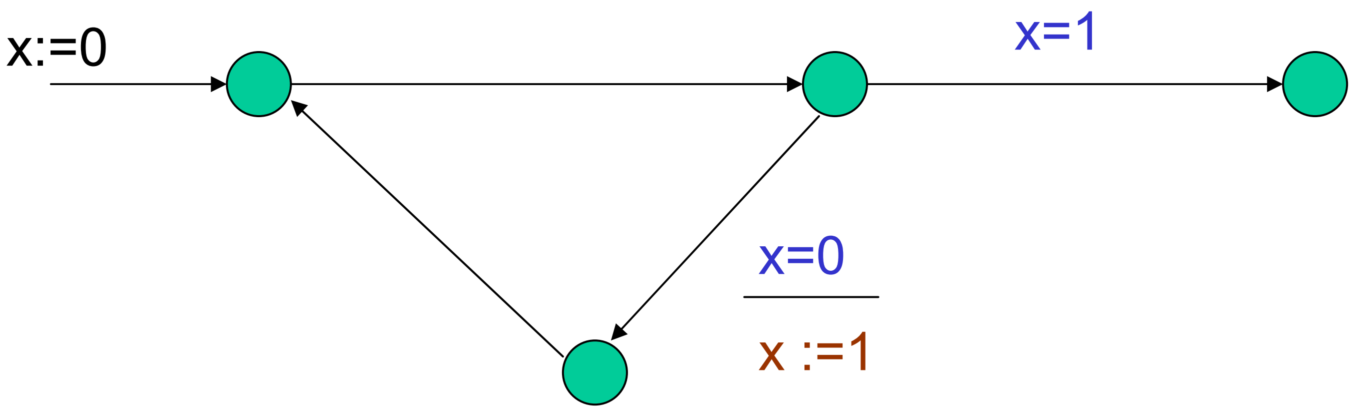


- Initial state, variable assignment

Covering Tests for EFSM

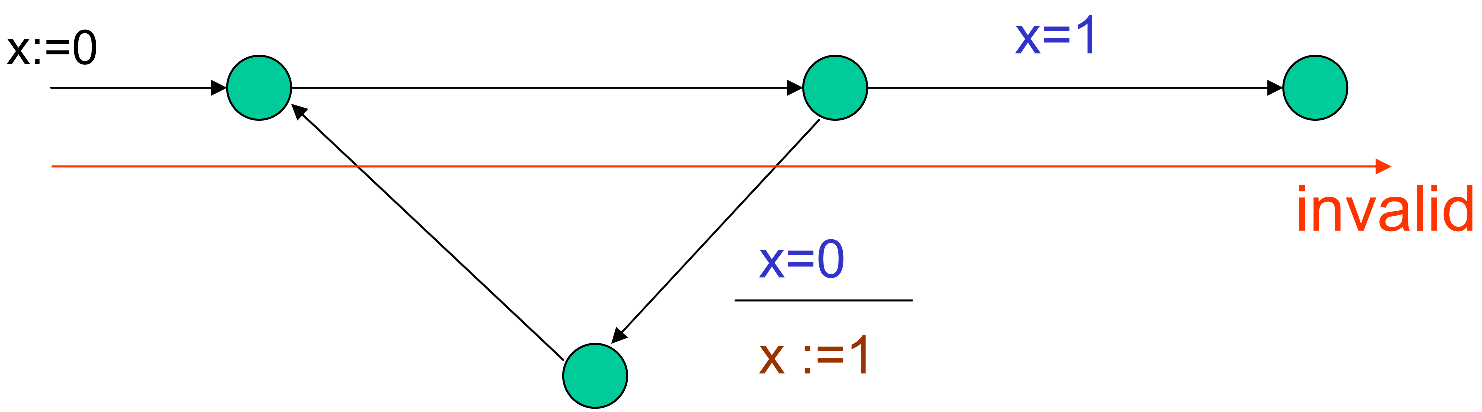
- Find minimum number of valid paths that cover all the transitions of the EFSM

  (or all states, or all transitions + conditions, ….)



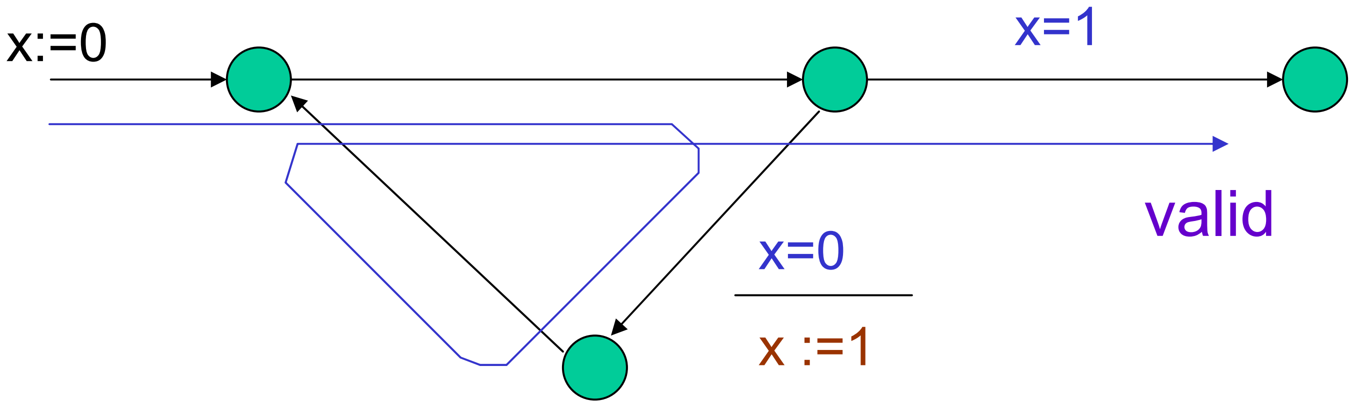
Covering Tests for EFSM

- Find minimum number of valid paths that cover all the transitions of the EFSM
- Not all paths of the EFSM graph are valid



Covering Tests for EFSM

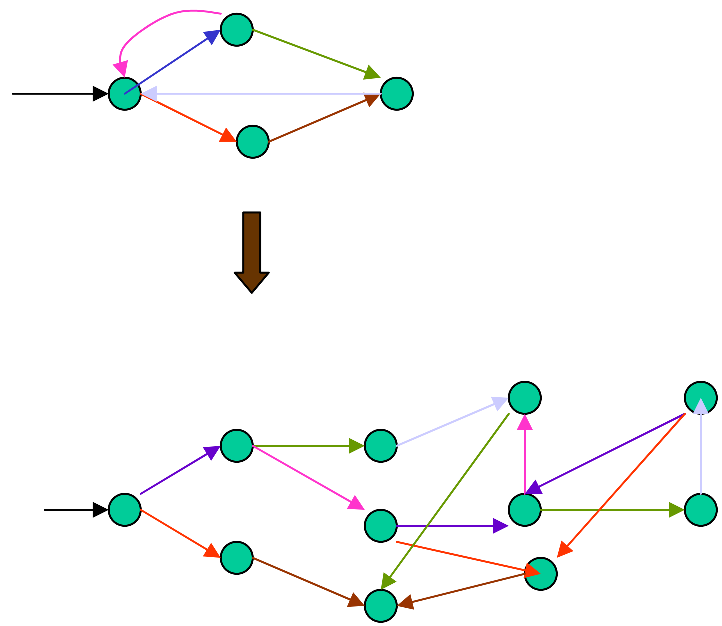
- Find minimum number of valid paths that cover all the transitions of the EFSM
- Not all paths of the EFSM graph are valid



EFSM → Colored Graph

- One color per transition of EFSM

EFSM



Expanded FSM

(no variables)

Find minimum number of paths covering all the colors

EFSM $\rightarrow$ Colored Graph

- The full expanded graph may be too large

- We may generate a subgraph of it
  - ok: all paths in the subgraph are valid
  (but subgraph may lack some other potentially better paths)

- We can generate a *minimized graph*
  (equivalence relation on expanded states that respects transitions and colors)
  - does not miss any paths

Colored Graph Covering Problem

- Given a graph with colored edges (and/or nodes), find minimum set of paths covering all colors

- Reduces to acyclic graphs:

  compute strongly connected components and shrink them

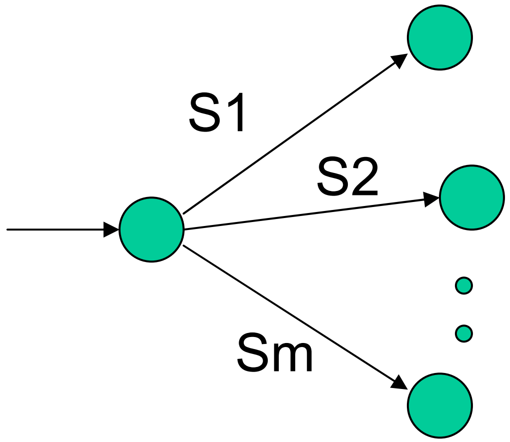
- Dynamic programming algorithm:

  exponential in #colors.

- In general, problem is NP-hard.
- At least as hard as Set Cover problem $\Rightarrow$ cannot approximate min # paths within $< \log n$ factor.

Relation to Set Cover

- Set Cover Problem:
  - Given a set U of elements and a family F of subsets S1, …, Sm of U, find minimum number of sets from F that covers U
- Colored graph has color set = U



Greedy Covering algorithm

Given a graph with colored edges,

- Find a path covering maximum number of colors
- Include the path in the solution, discard the colors and iterate

- The Max Color Path problem is also NP-hard and MAX SNP-hard

$\Rightarrow$ can't approximate below some constant factor

Max Color Path Problem

- Open: Is there a constant factor approximation algorithm?
- Simple Greedy $\rightarrow$ worst-case ratio = #colors
- "Transitive" (more careful) Greedy $\rightarrow$ finds path with at least $\sqrt{OPT}$ colors, if best path has *OPT* colors
- Another quasipolynomial algorithm finds path with at least *OPT* / c·log(*OPT*) colors

Min Covering Problem

- Often in practice, the first few paths of the Greedy covering algorithm cover many colors, and then the rest of the paths, which form the bulk of the test set, cover a few colors each

» In the latter part we can find optimal paths:

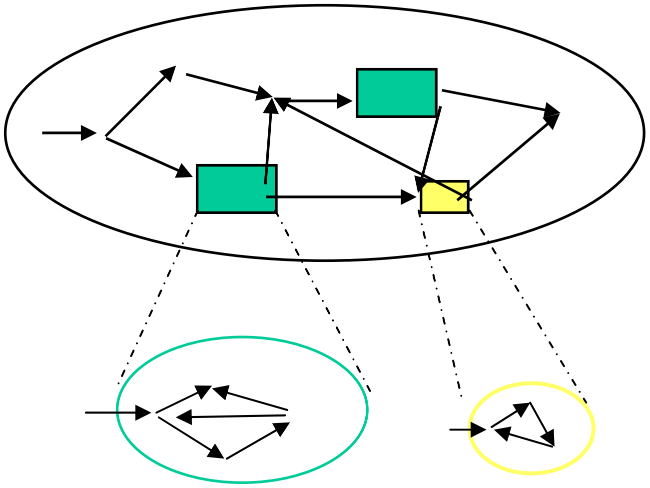
- Finding a path covering at least k colors with k fixed (or up to logn) can be done in polynomial time.

Pythia

- Toolset for automated test generation for FSM's and EFSM's (Lee & Yannakakis)
- Incorporates optimization algorithms
- Applications to systems:
  - PHS, 5ESS INAP, Diamond, H.248

Hierarchical FSM/Graph

Nodes are ordinary states or superstates mapped to lower level FSMs/graphs



Compact representation of large flat FSM

- Useful way to structure large FSM

Hierarchical Testing

- Approach 1: Expand the hierarchy

 - Cover every transition/state of every module (subgraph) in *every* possible context: Polynomial time in the flattened graph but the flat graph can be exponentially larger

 $\Rightarrow$ state explosion

 $\Rightarrow$ test explosion

- Approach 2: No expansion

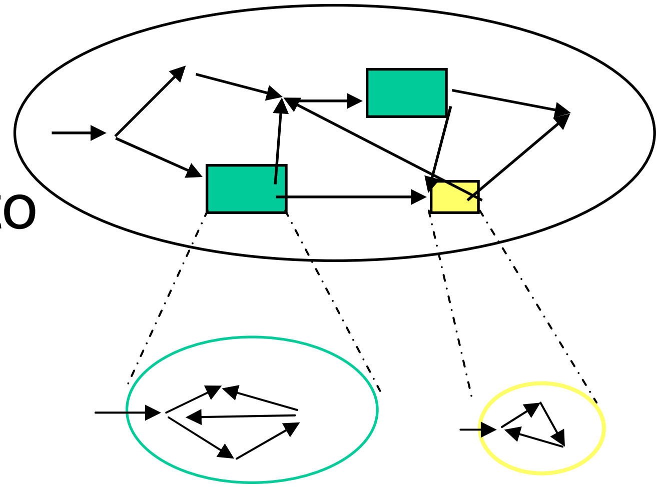
 - Cover every transition/state in at least *some* context,

- Find minimum number of tests to cover all transitions of all the modules

Goal: construct the tests without expanding the hierarchy

Hierarchical FSM Testing

- Find minimum number of tests to cover all transitions of all the modules



- Could expand to flat FSM and reduce to the colored graph covering problem

- Much better: Can avoid flattening (state explosion) and can get constant approximation ratio = nesting depth [MY]

- Can also approximate more general metrics that involve both the number of paths and their length, costs on the edges, …

Conclusions

- Long line of research
- Theoretical and practical interest
- Rich variety of problems
- Connections with different areas (optimization, verification, learning, games, combinatorics,…)