SYSC 4101 / 5105

Graph Criteria—Applications
Functional–State Based

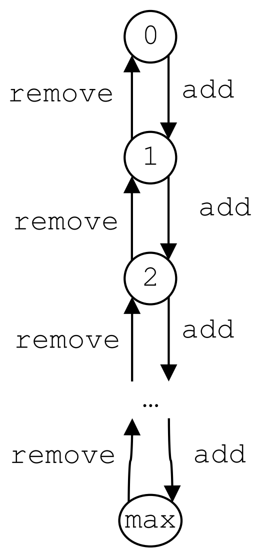
FSM, EFSM, UML State Machine

- Finite State Machine [Ammann & Offutt]
  - Nodes are states and edges are transitions
- State
  - A state represents a recognizable situation that remains in existence over some period of time
  - A state is defined by specific values for a set of variables; as long as those variables have those values, the software is considered to be in that state
- Transition
  - A transition is thought of as occurring in zero time and usually represents a change to the values of one or more (state) variables
  - When the variables change, the software is considered to move from the transition’s pre-state to the transition’s post-state
  - If a transition’s pre-state and post-state are the same then values of the state variables will not change

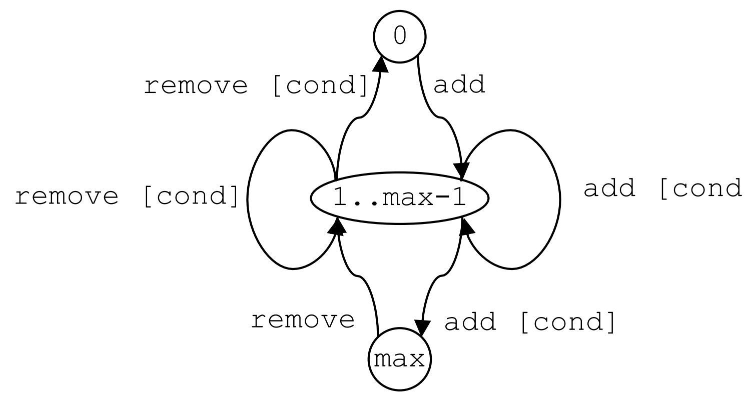
➢ A discussion of concrete states (FSM view) rather than abstract states (UML state machine view)

Concrete State vs. Abstract State

- Concrete State
  - Specific values for a set of (state) variables
- Example: a bounded bag (with at most max elements)



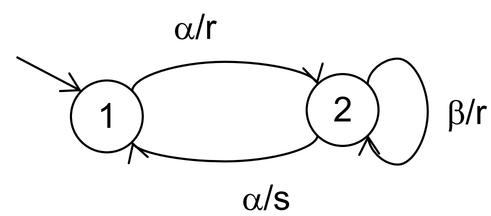
- Abstract State
  - Specific sets of values for a set of (state) variables
- Example: a bounded bag (with at most max elements)



Finite State Machines (I)

For a sequential functionality

S : finite set of states
S_0 : initial state
∑ : finite input alphabet
Ω : finite output alphabet
δ : transition function, $\delta: S \times \Sigma \rightarrow S$
λ : characterization function, $\lambda: S \times \Sigma \rightarrow \Omega$



Graphical representation

state	α	β
1	2/r	?
2	1/s	2/r

Tabular representation

First steps:

- Verify completeness, e.g., add a transition ‘ β/- ’ which loops on 1
- Deterministic ? Fully specified? Minimal ? Strongly connected ?

Graph Criteria → State Machine Criteria

- All Nodes = All States
- All Edges = All Transitions
- All Edge-pairs = All Transition-pairs
- Prime path
- Simple round trip
- Complete round trip
- Complete path
- All-Defs
- All-Uses
- All-DU paths

→ Transition tree

→ This is still research

Issues:

- What is a definition or use is not clear?
- How to identify them (simply from state machine, from accompanying contracts)?

Controllability and Observability

- Controllability
  - How to reach a state from where we can trigger a test requirement (e.g., state, transition, round trip path)?
- Observability
  - How to evaluate whether a test case succeeds (oracle)?
- Built-in support
  - setState(), getState() operations
  - Checking pre, post conditions and invariants (oracle)
- Using the model
  - Finding a transition sequence from the initial state to the state to be reached before triggering the test requirement (the W-method—see later)
  - Finding a transition sequence to trigger once we have triggered the test requirement

State-Based Testing

- Suppose a test requirement of the form: exercise transition from state i to state j (i.e., $i \rightarrow j$ or e_{ij}):
- General strategy:
  1. Need to reach state i
  2. Need to trigger e_{ij}
  3. Need to collect data for oracle
- Generating test sequence (transition sequence) T_{ij} of the form

  T_{ij} = $preamble_i \bullet e_{ij} \bullet SC_j$

  Where :

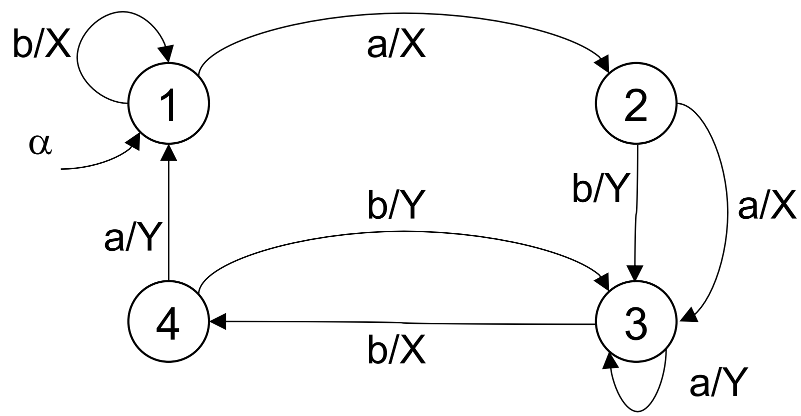
  - $preamble_i$ = sub-path starting from the initial state of state machine and arriving at state i (simplest case: reset function)
  - e_{ij} = transition $i \rightarrow j$
  - SC_j = a sub-path that can be used to check we have reach state j

State-Based Testing (cont.)

- Sub-path for oracle?
- Characterization set (or characterization sequence), named W
  - Observing the sequence of outputs produced by the characterization sequence execution allows us to deduce the original state to which it (the characterization sequence) was applied (in the simplest case, a status function)
  - The characterization sequence is unique (produced outputs) to the state to which it is applied.
- Other techniques exist

State-Based Testing (cont.)

- Characterization set (or characterization sequence) example



	a	b	aa	ab	ba	bb
1	X	X	XX	XY	XX	XX
2	X	Y	XY	XX	YY	YX
3	Y	X	YY	YX	XY	XY
4	Y	Y	YX	YX	YY	YX

In state 1:
- Input a produces X
- Input b produces X
- Input aa produces XX
- …

- Characterization sequence(s)?
  - Not {a}: cannot distinguish 1 from 2
  - Not {b}: cannot distinguish 1 from 3
  - Not {ab}: cannot distinguish 3 from 4
  - Not {ba}: cannot distinguish 2 from 4
  - Not {bb}: cannot distinguish 2 from 4
  - {aa} is one: output response is unique to the state

Chow's Methods for State Model Testing

One of the earliest papers on the topic is Chow's paper (1978)

- Does not address guard conditions on transitions (operations and actions only)

Chow's approach (adapted by Binder):

- The first step is to generate a *transition or test tree* from the state machine
  - In graph theory this is also called a spanning tree
  - Assumes a unique initial state
    - If more than one initial state, create a pseudo unique initial state that leads (transitions) to the "real" initial states
- The tree paths include:
  - All *round-trip* paths (as defined by Binder)
  - And *simple paths* from the initial to the final state of the state model
- Append each characterization sequence (W) or call 'getState()'

Procedure for Deriving Tree

1. The initial state is the root node of the tree, mark node as non-terminal
2. Examine the state that corresponds to each non-terminal node in the tree and each outgoing transitions from this state
   a. Draw one branch in the tree for each of the outgoing transitions
3. For each edge and node drawn in step 2:
   a. Note the corresponding state transition information on the branch
   b. If the state that the new node represents is already represented by another node that has outgoing edges (in the tree) anywhere in the diagram, or is a final state, mark this node as terminal.
      i.e., no more transitions are drawn from a terminal node
4. Repeat steps 2 and 3 until all leaf nodes are marked terminal

This procedure corresponds to a breadth-first search

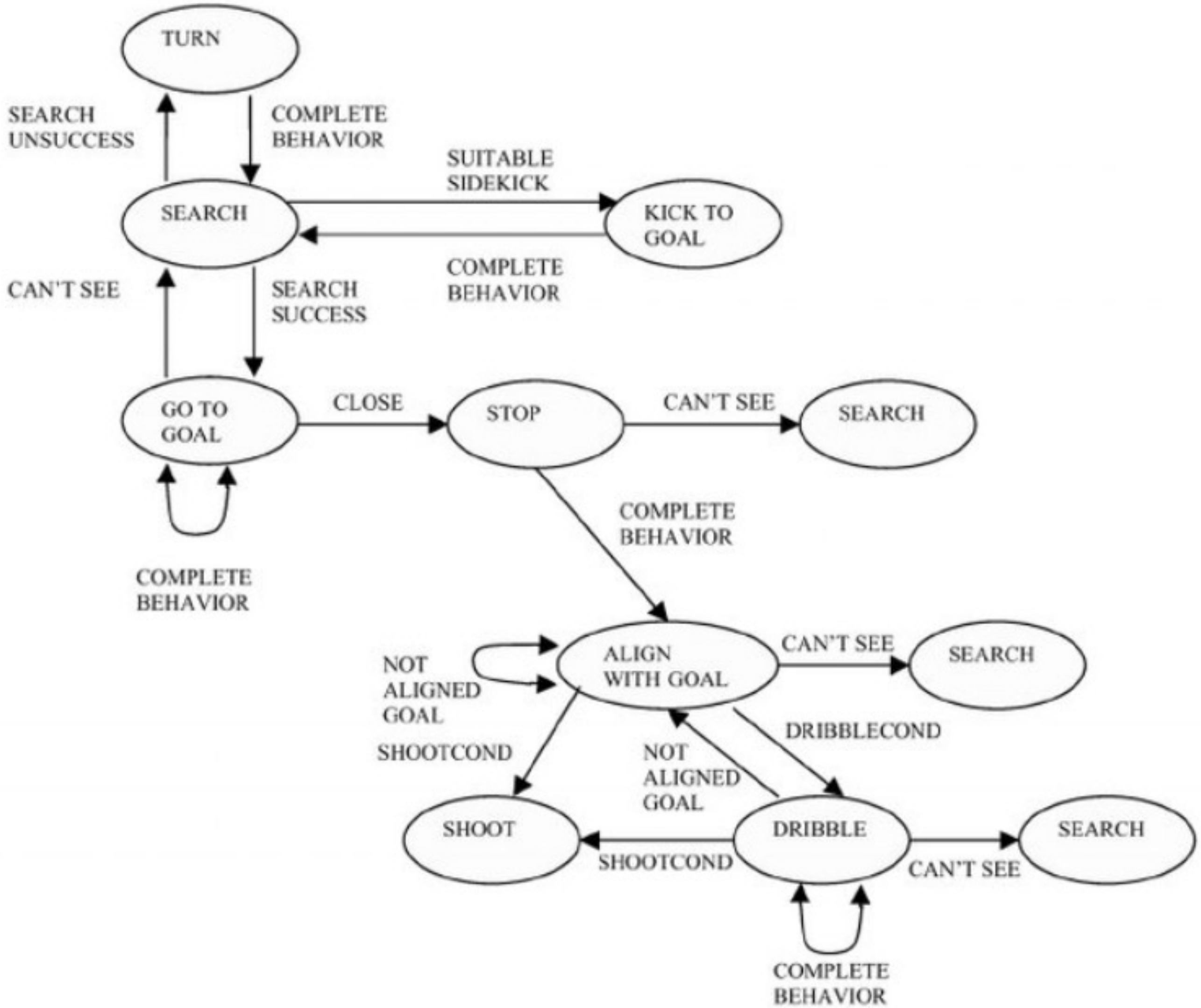
- A depth first search yield fewer, longer test sequences

The order in which states and transitions are investigated is supposed to be irrelevant

From Test Tree to Test Cases

- Each test case begins at the root node and ends at a leaf node
- The expected result (Oracle) is the sequence of *states* and *actions* (outputs, other objects' change of state)
- Test cases are completed by identifying *method parameter values* and *required conditions* to traverse a path
- We run the test cases by setting the object under test to the *initial state*, applying the test sequence, and then checking the *intermediary* states, *final* state and outputs (e.g., logged)

Example


TURN
SEARCH UNSUCCESS
COMPLETE BEHAVIOR
SUITABLE SIDEKICK
SEARCH
KICK TO GOAL
COMPLETE BEHAVIOR
CAN'T SEE
SEARCH SUCCESS
GO TO GOAL
CLOSE
STOP
CAN'T SEE
SEARCH
COMPLETE BEHAVIOR
COMPLETE BEHAVIOR
NOT ALIGNED GOAL
ALIGN WITH GOAL
CAN'T SEE
SEARCH
DRIBBLECOND
SHOOTCOND
NOT ALIGNED GOAL
SHOOT
DRIBBLE
SEARCH
SHOOTCOND
CAN'T SEE
COMPLETE BEHAVIOR

Guard Conditions

- Binder adapted Chow’s strategy to EFSMs.
- Step 2 in the previous algorithm is modified:
  a. If the transition is unguarded, draw one new branch
  b. If the transition guard is a simple Boolean expression or contains only logical `AND` operators (only one way to make it true), draw one new branch (true combination)
  c. If the transition guard is a complex Boolean expression using one or more `OR` operators (several ways to make it true), draw a new branch for each truth value combination that makes the guard true.
- Another adaptation when the guard specifies a relationship that occurs only after repeating some event
  – Draw a single arc annotated with * for the transition
  – Cannot be performed automatically (test engineer)

Incomplete State Machine?

- Example 1:
  - Suppose the input alphabet is {a,b}, state ‘1’ has an outgoing transition for input ‘a’ but does not have an outgoing transition for input ‘b’.
  - State machine does not specify what happens when input ‘b’ is received in state ‘1’
    - Assumption is that input ‘b’ is ignored when in state ‘1’.
- Example 2:
  - Suppose the input alphabet is {a,b}, state ‘1’ has an outgoing transition for input ‘a’ with guard ”var=12” and an outgoing transition for input ‘b’.
  - State machine does not specify what happens when input ‘a’ is received and the guard is false.
    - Assumption is that input ‘a’ is ignored when in state ‘1’ with false guard.

➢ Incomplete specifications

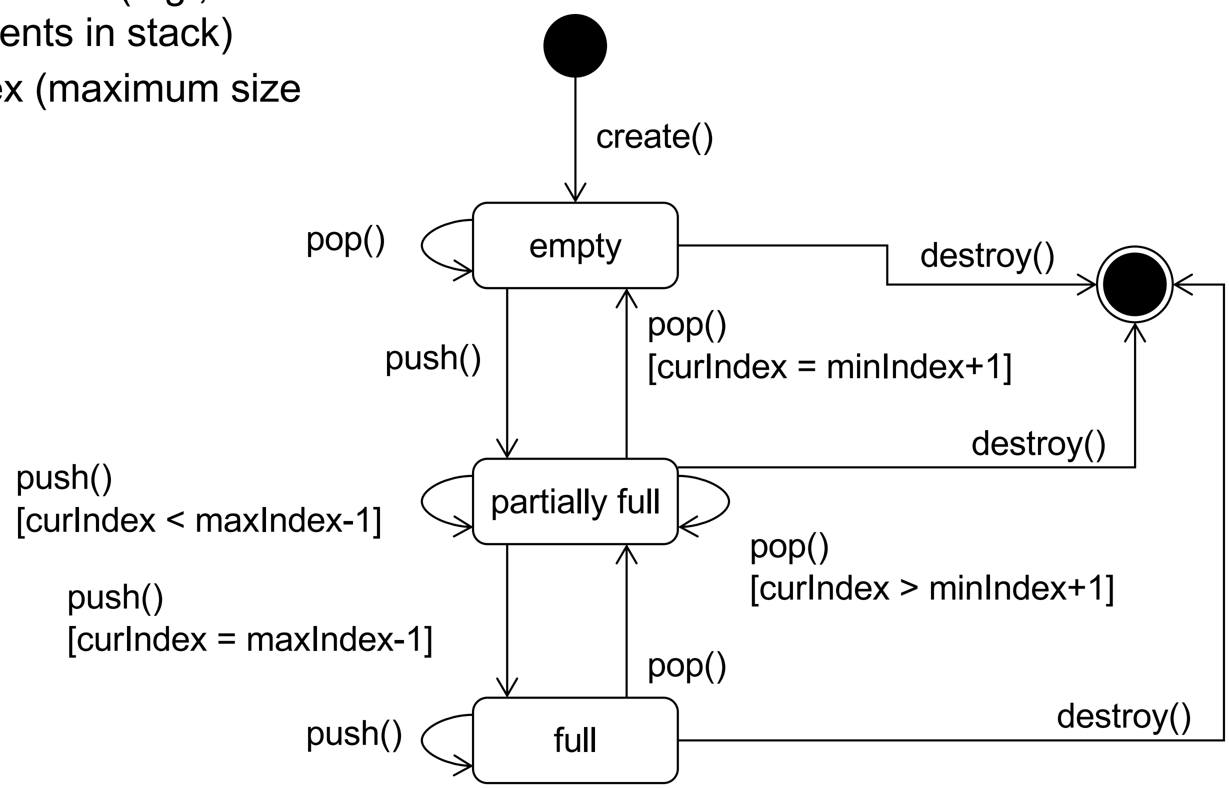
Incomplete State Machine? (cont.)

- Test case definition from an incomplete state machine?
  - We derive test paths from the state machine graph
  - If the state machine graph does not specify something, that behavior is not exercised!
- Binder calls for “sneak paths”
  - For each unspecified behavior
    - Input ‘b’ when in state ‘1’
    - Input ‘a’ when in state ‘1’ and the guard is false
  - Find a sub-path to go from initial state to unspecified behavior
  - Trigger unspecified behavior
  - Check that nothing has changed (e.g. with characterization sequence)
    - Recall the assumption is that input is ignored.

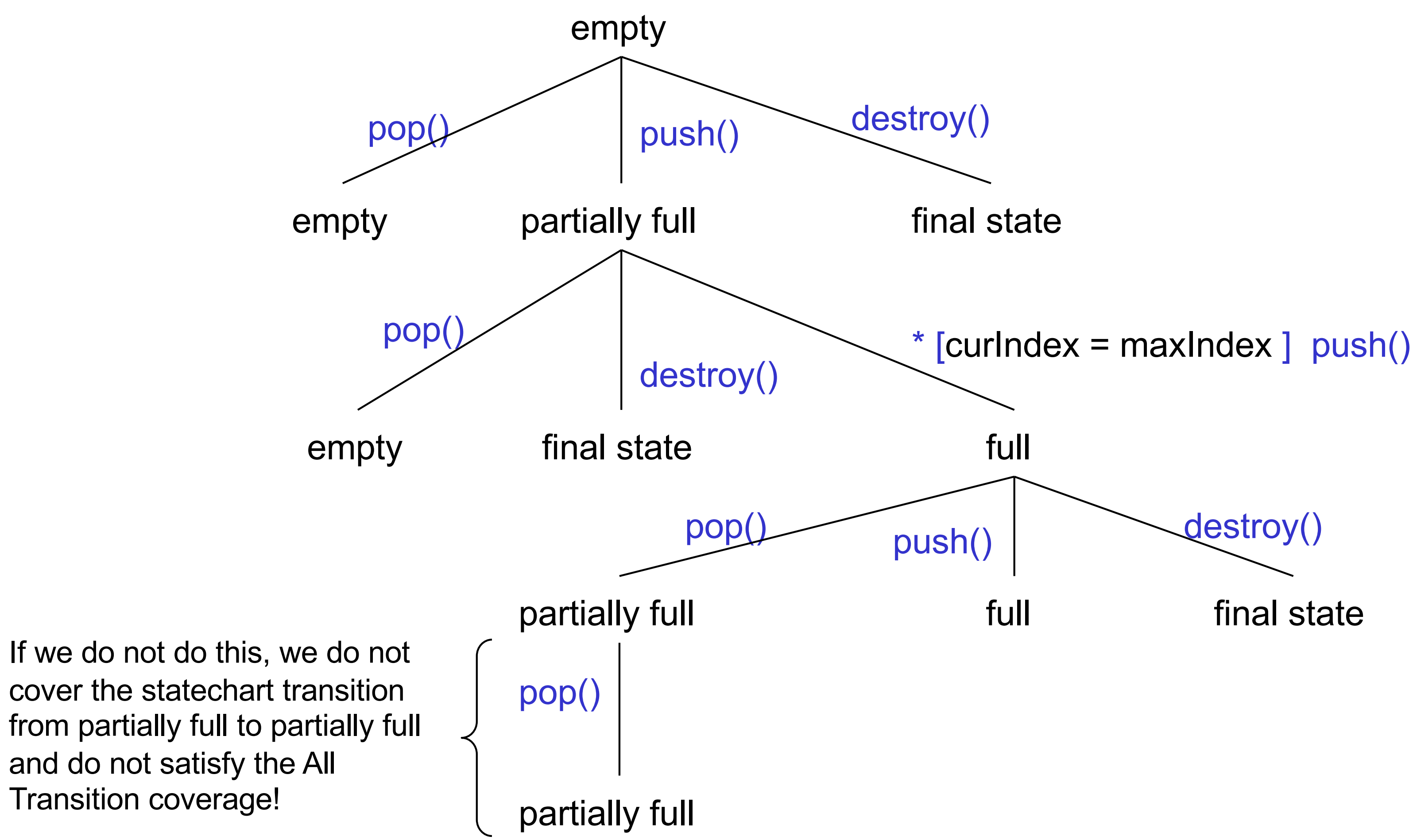
BoundedStack Example

Assume that three data members are defined in the class:

- curIndex: current index of last element introduced in the stack
- minIndex: minimum index value (e.g., 0 minimum number of elements in stack)
- maxIndex: maximum index (maximum size of the stack)



BoundedStack Transition Tree



Problems with Binder's Approach

- Covering certain transitions requires to traverse specific paths to satisfy the guard condition.
  - We cannot simply follow the test tree algorithm when the statechart contains guard conditions.
  - How do we automate the generation of the tree?
- Sometimes, as a result of the above problem, using the tree algorithm does not lead to covering all transitions!
- The test tree covers round trip paths in a piecewise manner
  - It does not execute the round trip paths per se.
  - Therefore, we cannot say that the round trip path technique subsumes the n-transitions sequence criterion
- The test tree may not exercise the main functionalities of the system