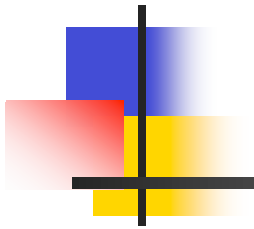
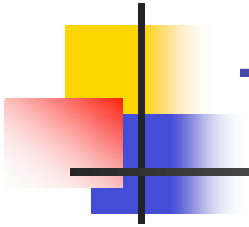


State-Based Testing
Part C – Test Cases

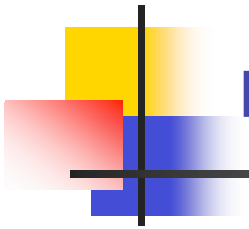
Generating test cases for complex behaviour

Reference: Robert V. Binder
Testing Object-Oriented Systems: Models, Patterns, and Tools
Addison-Wesley, 2000, Chapter 7



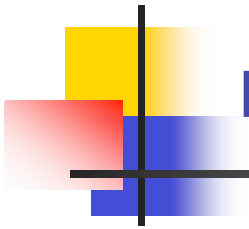
Test Strategies

- Exhaustive
- All Transitions
  - **Every transition executed at least once**
  - **Exercises all transitions, states and actions**
  - **Cannot show incorrect state is a result**
  - **Difficult to find sneak paths**
- All n-transition sequences
  - **Can find some incorrect and corrupt states**
- All round trip paths
  - **Generated by N+ test strategy**
    - **A prime path of nonzero length that starts and ends at the same node**



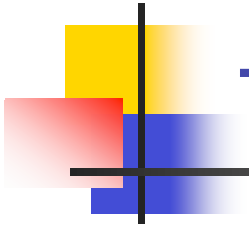
N+ Test Strategy Overview

- The N+ Test strategy
  - **Encompasses UML state models**
  - **Testing considerations unique to OO implementations**
  - **It uses a flattened model**
  - **All implicit transitions are exercised to reveal sneak paths**
  - **Relies on an the implementation to properly report resultant state**
  - **More powerful than simpler state-based strategies**
    - **Requires more analysis**
    - **Has larger test suites**
    - **Look at cost/benefit tradeoff**



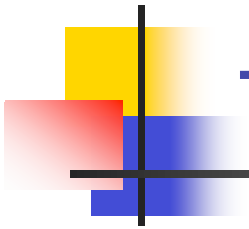
N+ Coverage

- N+ coverage reveals
  - **All state control faults**
  - **All sneak paths**
  - **Many corrupt state bugs**
  - **Many super-class/sub-class integration bugs**
  - **If more than one from α transition, reveals faults on each one**
  - **All transitions to the ω states**
  - **Can suggest presence of trap doors when used with program text coverage analyzer**



The N+ Test Strategy Development

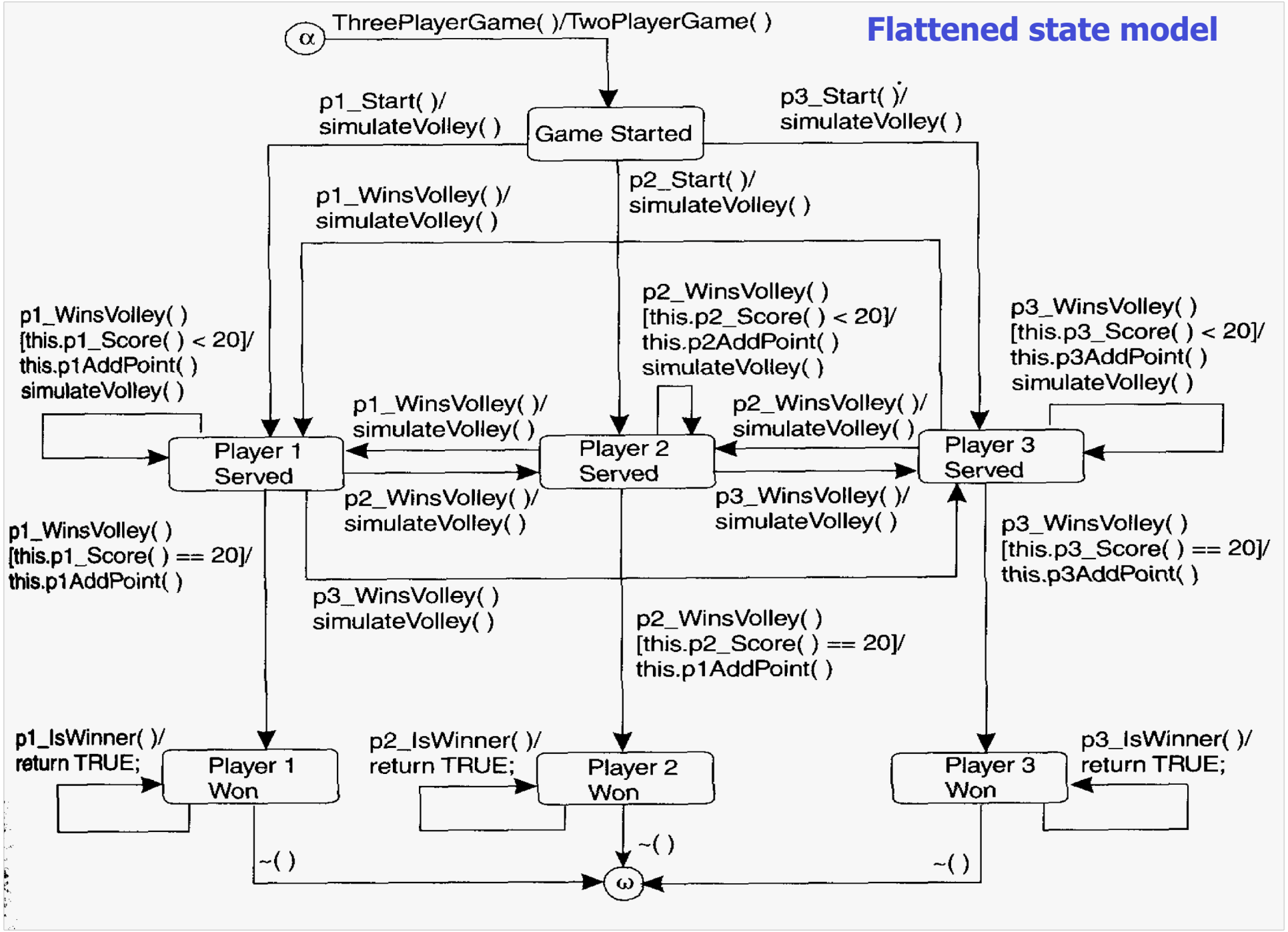
- Develop a state-based model of the system
  - **Validate the model using the checklists**
  - **Flatten the model – Expand the statechart**
  - **Develop the response matrix**
- Generate the round-trip path tree
- Generate the round-trip path test cases
- Generate the sneak path test cases
- Sensitize the transitions in each test case
  - **Find input values to satisfy guards for the transitions in the event path**
  - **Similar to finding path conditions in path testing**



The 3-player game example

- We will use an extension of the 2-player game as an example
- There is now a third player that may win any of the volleys

Flattened state model

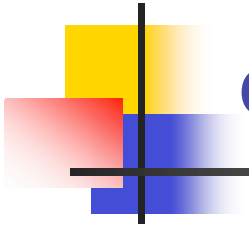


Response Matrix

Events and Guards			Accepting State/Expected Response								
			α	Game Started	Player 1 Served	Player 2 Served	Player 3 Served	Player 1 Won	Player 2 Won	Player 3 Won	ω
ctor			✓	6	6	6	6	6	6	6	6
p1_Start				✓	4	4	4	4	4	4	6
p2_Start				✓	4	4	4	4	4	4	6
p3_Start				✓	4	4	4	4	4	4	6
p1_WinsVolley	p1_score < 20	p1_Score == 20									
	DC	DC		4		✓	✓	4	4	4	6
	F	F			6						
	F	T			✓						
	T	F			✓						
	T	T									
p2_WinsVolley	p2_score < 20	p2_Score == 20									
	DC	DC		4	✓		✓	4	4	4	6
	F	F				6					
	F	T				✓					
	T	F				✓					
	T	T									
p3_WinsVolley	p3_score < 20	p3_Score == 20									
	DC	DC		4	✓	✓		4	4	4	6
	F	F					6				
	F	T					✓				
	T	F					✓				
	T	T									
p1_isWinner				✓	✓	✓	✓	✓	✓	✓	6
p2_isWinner				✓	✓	✓	✓	✓	✓	✓	6
p3_isWinner				✓	✓	✓	✓	✓	✓	✓	6
Other Public Accessors				✓	✓	✓	✓	✓	✓	✓	6
dtor				✓	✓	✓	✓	✓	✓	✓	6

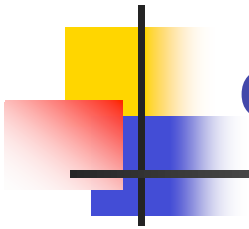
Generate Round-Trip Path Tree (GRTPT)

- Root
  - **Initial state – use α state with multiple constructors**
- First edges
  - **Draw for each transition out of initial state and add node for resultant state**
- Remaining edges
  - **Draw for each transition out of a leaf node and add node for resultant state**
  - **Mark new leaf nodes as terminal nodes, if new leaf is**
    - **Already in the tree**
    - **A final state**
    - **An ω state**



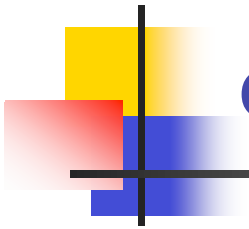
GRTPT– Traversing the FSM

- Use either a breadth-first or depth-first strategy for traversing the FSM
- Breadth-first
  - **Many short test sequences**
- Depth-first
  - **Fewer long test sequences**



GRTPT – Guarded Transitions

- At least two test conditions are necessary
- Model each true condition
  - **If several conditional variants can make a guard true, transcribe one transition for each variant**
    - **Guard is a simple Boolean expression, or contains only logical "and"**
      - Then only one transition is needed
    - **Guard is compound Boolean expression, has at least one logical "or" operator**
      - Then one transition is required for each predicate combination that yields a true result

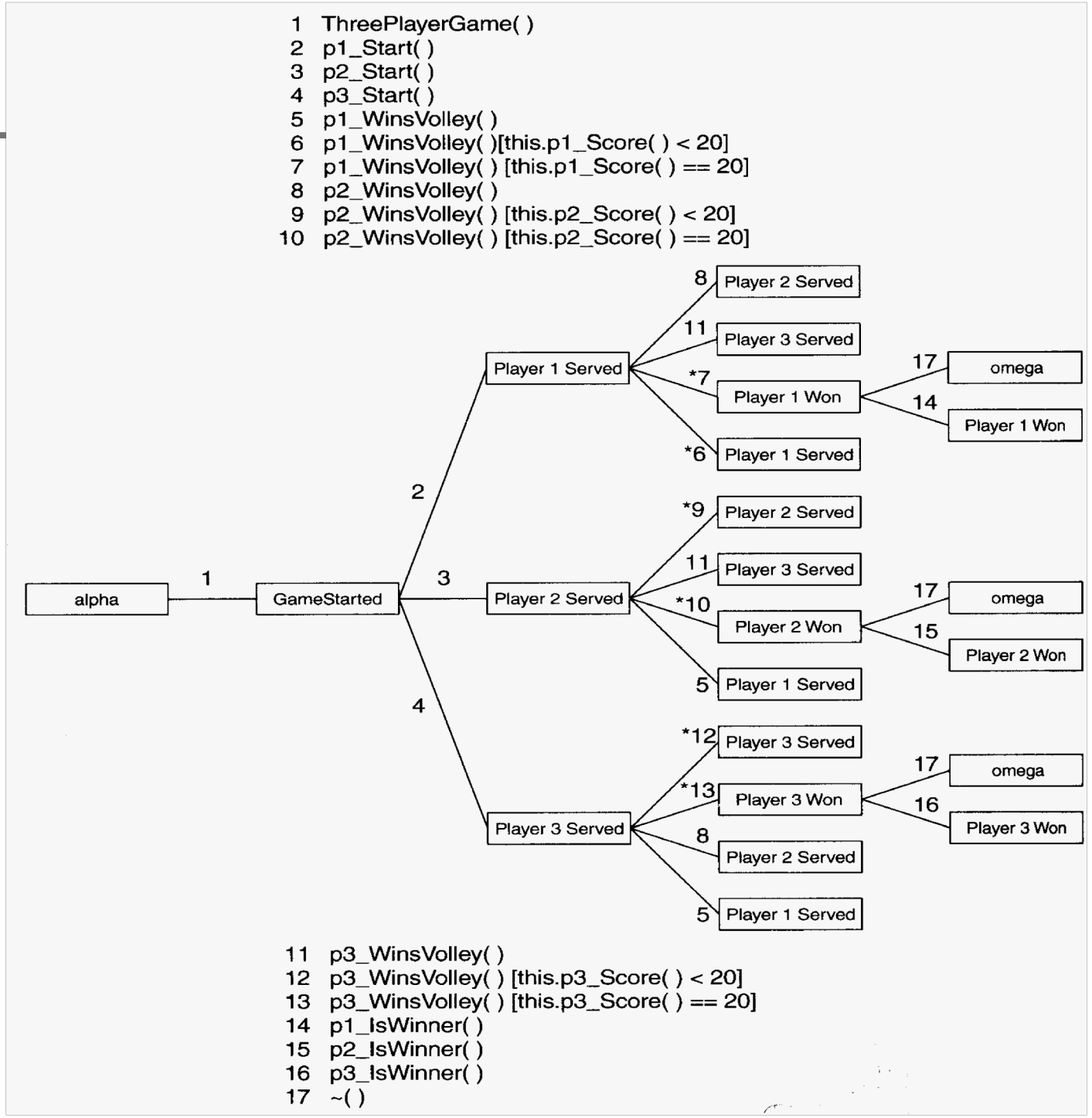


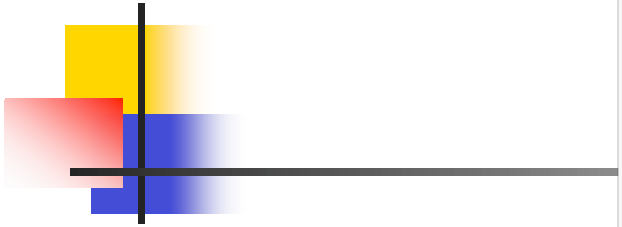
GRTPT – Guarded Transitions – 2

- **Guard specifies a relationship that occurs only afer repeating some event, [counter ≥ 10]**
  - Test sequence requires at least the number of iterations to satisfy the condition. The transition is graphed with a single arc annotated with an asterisk.

- **Test at least one false combination**
- **Tests to cover each guard's false variants are developed for the sneak attack tests**
  - **Recall variant testing for decision tables – there are others as well**

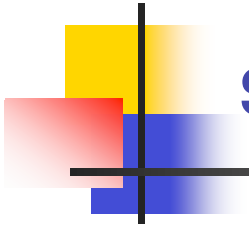
Transition tree for the 3-player game





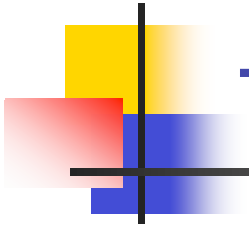
Generated test cases

	Test Case Input		Expected Result	
TCID	Event	Test Condition	Action	State
1.1	ThreePlayerGame			GameStarted
1.2	p1_start		simulateVolley	Player 1 Served
1.3	p2_WinsVolley		simulateVolley	Player 2 Served
2.1	ThreePlayerGame			GameStarted
2.2	p1_start		simulateVolley	Player 1 Served
2.3	p3_WinsVolley		simulateVolley	Player 3 Served
3.1	ThreePlayerGame			GameStarted
3.2	p1_start		simulateVolley	Player 1 Served
3.3	*		*	Player 1 Served
3.4	p1_WinsVolley	p1_Score == 20		Player 1 Won
3.5	dtor			omega
4.1	ThreePlayerGame			GameStarted
4.2	p1_start		simulateVolley	Player 1 Served
4.3	*		*	Player 1 Served
4.4	p1_WinsVolley	p1_Score == 20		Player 1 Won
4.5	p1_IsWinner		return TRUE	Player 1 Won
5.1	ThreePlayerGame			GameStarted
5.2	p1_start		simulateVolley	Player 1 Served
5.3	*		*	Player 1 Served
5.4	p1_WinsVolley	p1_Score == 19	simulateVolley	Player 1 Served
6.1	ThreePlayerGame			GameStarted
6.2	p2_start		simulateVolley	Player 2 Served
6.3	*		*	Player 2 Served
6.4	p2_WinsVolley	p2_Score == 19	simulateVolley	Player 2 Served
7.1	ThreePlayerGame			GameStarted
7.2	p2_start		simulateVolley	Player 2 Served
7.3	p3_WinsVolley		simulateVolley	Player 3 Served
8.1	ThreePlayerGame			GameStarted
8.2	p2_start		simulateVolley	Player 2 Served
8.3	*		*	Player 2 Served
8.4	p2_WinsVolley	p2_Score == 20		Player 2 Won
8.5	dtor			omega



Sneak path testing

- Look for Illegal transitions and evading guards
- Transition tree tests explicit behaviour
- We need to test each state's illegal events
- A test case for each non-checked, non-excluded transition cell in the response matrix
- Confirm that the actual response matches the specified response

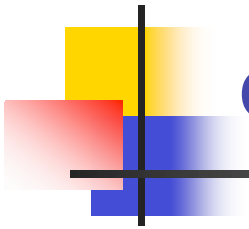


Testing one sneak path

- Put IUT (Implementation Under Test) into the corresponding state
  - **May need to have a special built-in test method, as getting there may take too long or be unstable**
  - **Can use any debugged test sequences that reach the state**
    - **Be careful if there are changes in the test suite**
- Apply the illegal event by sending a message or forcing the virtual machine to generate the desired event
- Check that the actual response matches the specified response
- Check that the resultant state is unchanged
  - **Sometimes a new concrete state is acceptable**

Sneak Path Test Suite

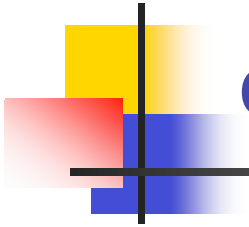
Test Case				Expected Result	
TCID	Setup Sequence	Test State	Test Event	Code	Action
16.0	ThreePlayerGame	Game Started	ThreePlayerGame	6	Abend
17.0	ThreePlayerGame	Game Started	p1_WinsVolley	4	IllegalEventException
18.0	ThreePlayerGame	Game Started	p2_WinsVolley	4	IllegalEventException
19.0	ThreePlayerGame	Game Started	p3_WinsVolley	4	IllegalEventException
20.0	10.0	Player 1 Served	ThreePlayerGame	6	Abend
21.0	5.0	Player 1 Served	p1_start	4	IllegalEventException
22.0	10.0	Player 1 Served	p2_start	4	IllegalEventException
23.0	5.0	Player 1 Served	p3_start	4	IllegalEventException
24.0	1.0	Player 2 Served	ThreePlayerGame	6	Abend
25.0	6.0	Player 2 Served	p1_start	4	IllegalEventException
26.0	1.0	Player 2 Served	p2_start	4	IllegalEventException
27.0	6.0	Player 2 Served	p3_start	4	IllegalEventException
28.0	7.0	Player 3 Served	ThreePlayerGame	6	Abend
29.0	2.0	Player 3 Served	p1_start	4	IllegalEventException



Checking Resultant state

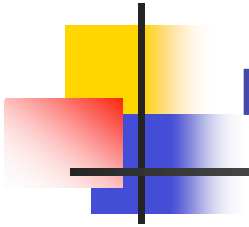
- State reporter
  - Can evaluate state invariant to determine state of object
  - Implement assertion functions

    bool isGameStarted() { … }
    - After each event appropriate state reporter is asserted
- Test repetition – good for corrupt states
  - Repeat test and compare results
  - Corrupt states may not give the same result
  - Not as reliable as state reporter method



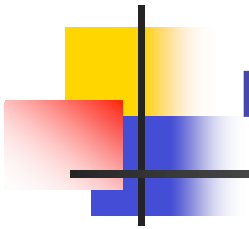
Checking Resultant state – 2

- State revealing signatures
  - **Identify and determine a signature sequence**
    - **A sequence of output events that are unique for the state**
    - **Analyze specification**
  - **Expensive and difficult**



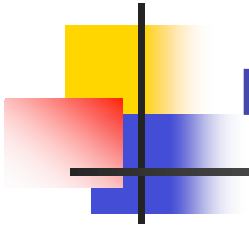
Major test strategies in increasing power

- Piecewise
  - **Every state, every event, every action at least once**
  - **Does not correspond to state model**
  - **Inadequate for testing**
- All transitions – minimum acceptable
  - **Every transition is exercised at least once**
  - **Implies all states, all events, all actions**
  - **Incorrect / Missing event / action pairs are guaranteed**
  - **Does not show incorrect state is a result**
  - **Unless completely specified, sneak paths are not found**



Major test strategies in increasing power – 2

- All transition k-tuples
  - **Exercise every transition sequence of k events at least once**
    - **1-tuple is equivalent to all transitions**
  - **Not necessarily all incorrect or corrupt states are found**
- All round-trip paths
  - **Called N+ coverage**
  - **Shortest trip is to loop back once to the same state**
  - **The longest trip depends upon the structure of the FSM**



Major test strategies in increasing power – 3

- Any sequence that goes beyond a round trip must be part of a sequence that belongs to another round trip
- Finds all incorrect or missing event/action pairs
- Can find some incorrect or invalid states
  - E.g. enter state that mimics correct behaviour for 10 events but becomes corrupt on the 11'th
- N+ strategy relies on state inspector

Major test strategies in increasing power – 4

- M-length signature
  - **Used for opaque systems – cannot determine current state**
  - **A state signature is used to determine the current state of the IUT**
    - **A sequence of output actions unique for the state**
    - **If the actual state signature is the expected one, then in the correct state**
  - **To find corrupt states, need to try sequence long enough to get beyond any possible number of corrupt states, which is guessed as being M**
- Exhaustive

Test Suite Size

TABLE 7.12 Size of State-based Test Suite by Strategy and Size of IUT

Representative IUT				Test Strategy: All Transitions		N+		W-Method	
	n	*k*		(1)	(2)	(1)	(2)	(3)	(4)
3Player	7	9	Test Cases	21	32	63	63	63	63
			Messages	21	32	147	284	441	3087
Small	10	15	Test Cases	50	75	150	150	150	150
			Messages	50	75	500	1125	1500	15000
Med	15	30	Test Cases	150	225	450	450	450	450
			Messages	150	225	2250	6750	6750	101250
Large	30	100	Test Cases	1000	1500	3000	3000	3000	3000
			Messages	1000	1500	30000	150000	90000	2700000

Key: n = number of states, k = number of events

(1) Assumes average number of events per test case = $k/2$.
(2) Assumes average number of events per test case = $k/3$.
(3) Worst case minimum, approximately $n^2 \times k$ [Chow 78].
(4) Worst case maximum, approximately $n^3 \times k$ [Chow 78].