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**

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 α state reveals faults on each one**
  - **All transitions to the ω states**
  - **Can suggest presence of trap doors when used with program text coverage analyzer**

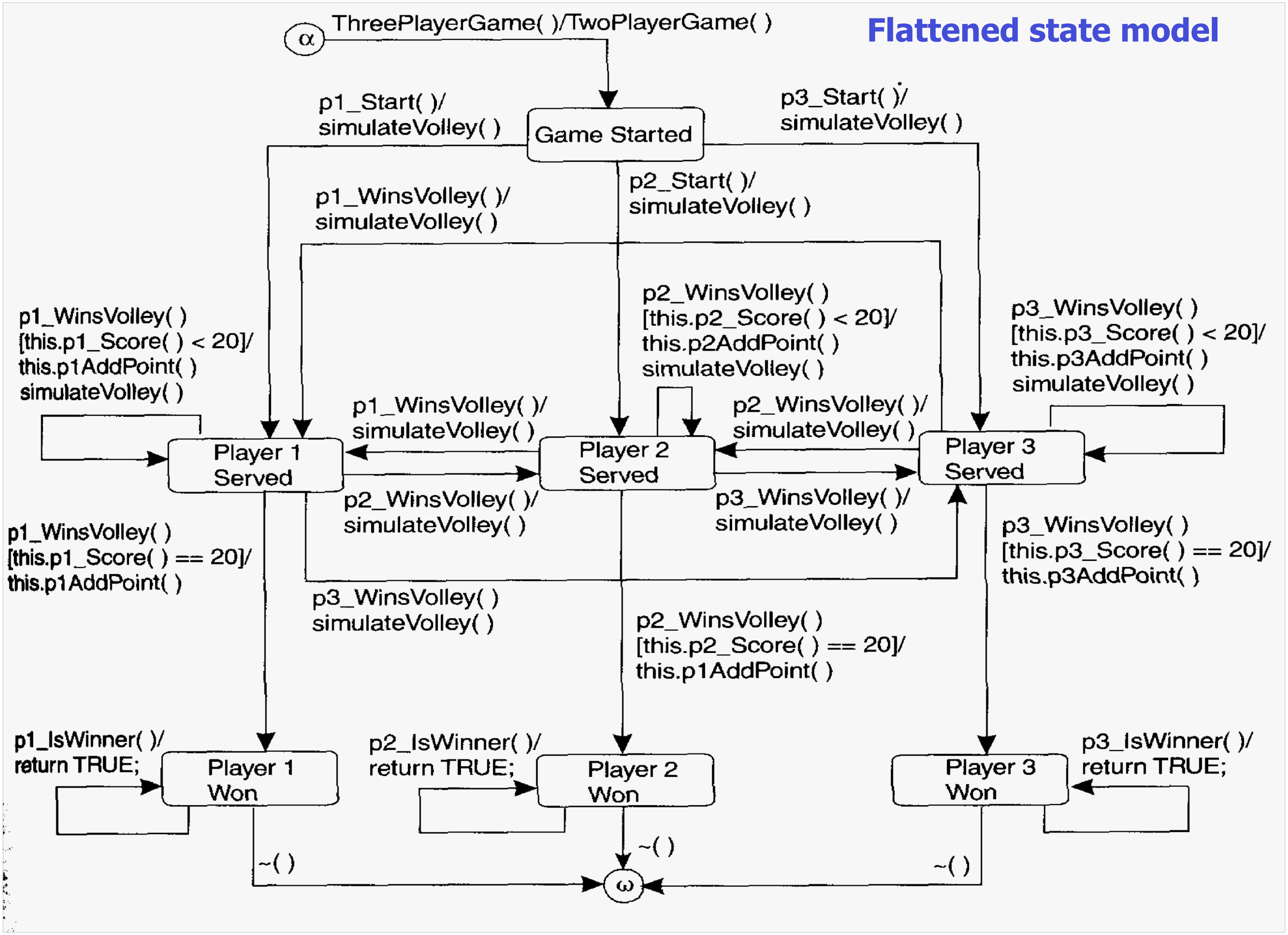
The N+ Test Strategy Development

- 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 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

Round-Trip Path Tree

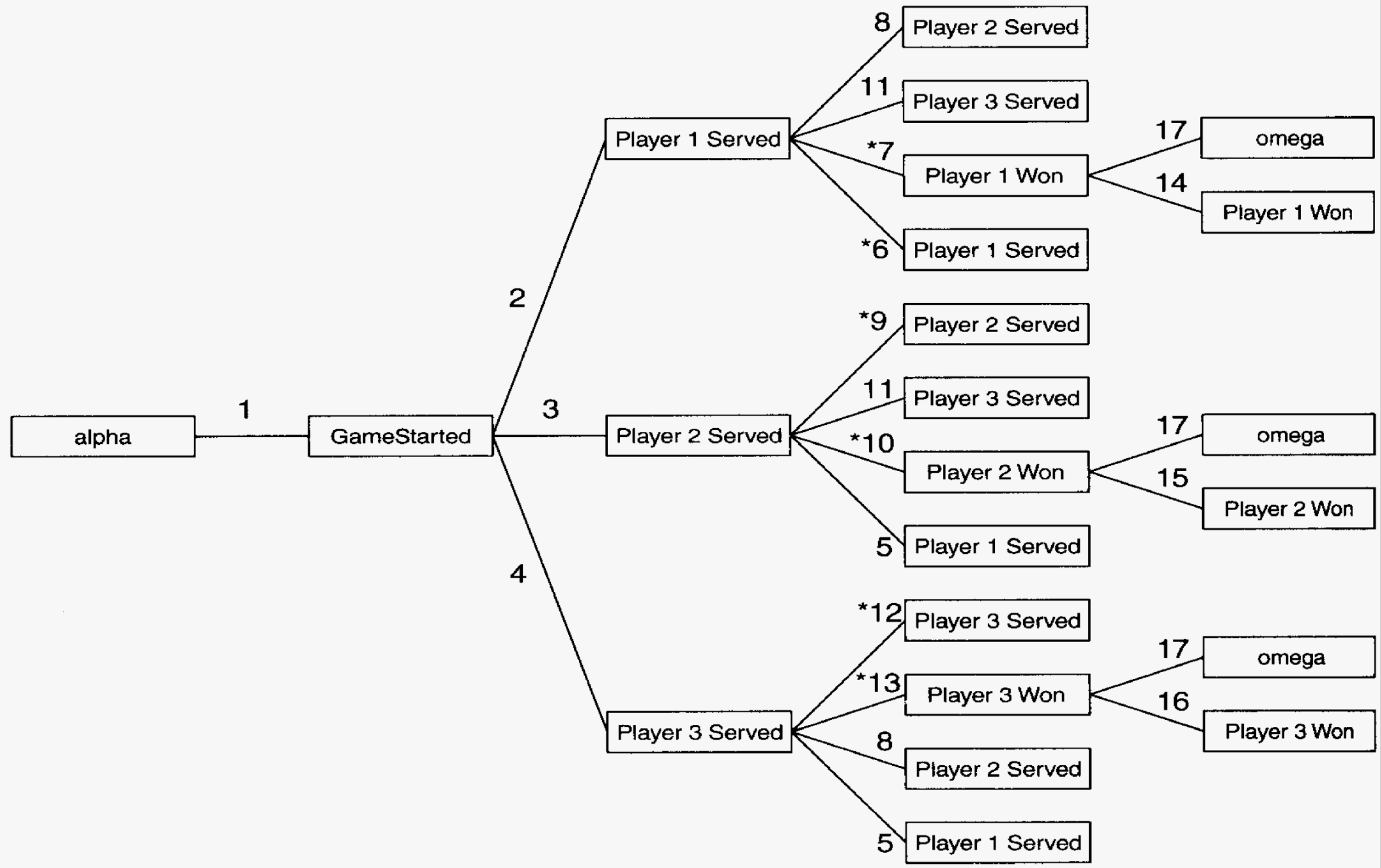
- Exercise all transitions and loops on every possible alpha-omega path at least once
- Root: Initial state – use α state with multiple constructors
- Edge for each transition
- Stop if the resultant state is already in the tree or is a final state

Round-Trip Path Tree – 2

- Guards
  - **One transition for each variant that evaluates to True**
    - **Simple Boolean expression containing only logical and – one test case**
    - **Compound expression containing at least one or – one test transition for each predicate combination giving true**
    - **Specifies a counter (counter ≥ 1000) – need to repeat transition until the count is satisfied**
  - **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

1 ThreePlayerGame( )
2 p1_Start( )
3 p2_Start( )
4 p3_Start( )
5 p1_WinsVolley( )
6 p1_WinsVolley( )[this.p1_Score( ) < 20]
7 p1_WinsVolley( ) [this.p1_Score( ) == 20]
8 p2_WinsVolley( )
9 p2_WinsVolley( ) [this.p2_Score( ) < 20]
10 p2_WinsVolley( ) [this.p2_Score( ) == 20]



11 p3_WinsVolley( )
12 p3_WinsVolley( ) [this.p3_Score( ) < 20]
13 p3_WinsVolley( ) [this.p3_Score( ) == 20]
14 p1_IsWinner( )
15 p2_IsWinner( )
16 p3_IsWinner( )
17 ~( )

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 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**
- 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**
- All transitions – minimum acceptable
  - **Every transition is exercised at least once**
- All transition k-tuples
  - **Exercise every transition sequence of k events at least once**
    - **1-tuple is equivalent to all transitions**
- All round-trip paths
  - **N+ coverage**
- M-length signature
  - **Used for opaque systems – cannot determine current state**
- Exhaustive