State-Based Testing
Part A – Modeling states

Generating test cases for complex behaviour

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

- We are interested in testing the behaviour of many different types of systems, including event-driven software systems.
- Interaction with GUI systems can follow a large number of paths.
- State machines can model event-driven behaviour.
- If we can express the system under test as a state machine, we can generate test cases for its behaviour.
OO Systems

- State-based testing is well suited to OO Systems
- Behaviour responsibility is distributed over
  - Classes, clusters, subsystem or system
  - Behaviour bugs due to complex and implicit structure
What is a state machine?
A state machine is …

- A system whose output is determined by both current state and past input
- Previous inputs are represented in the current state
- State-based behaviour
  - Identical inputs are not always accepted
    - Depends upon the state
  - When accepted, they may produce different outputs
    - Depends upon the state
What are the building blocks of a state machine?
Building blocks of a state machine – 2

- **State**
  - An abstraction that summarizes past inputs, and determines behaviour on subsequent inputs

- **Transition**
  - An allowable two-state sequence. Caused by an event

- **Event**
  - An input or a time interval

- **Action**
  - The output that follows an event
Describe the behaviour of a state machine?
State machine behaviour – 2

1. Begin in the initial state

2. Wait for an event

3. An event comes in
   1. If not accepted in the current state, ignore
   2. If accepted, a transition fires, output is produced (if any), the resultant state of the transition becomes the current state

4. Repeat from step 2 unless the current state is a final state
State machine properties

- How events are generated is not part of the model
- Transitions fire one at a time
- The machine can be in only one state at a time
- The current state cannot change except by a defined transition
- States, events, transitions, actions cannot be added during execution
State machine properties – 2

- Algorithms for output creation are not part of the model
- The firing of a transition does not consume any amount of time
  - An event is instantaneous
    - It has no beginning or ending
    - Beginnings and endings imply duration

The challenge
How to model the behaviour of a given system using a state machine?
State transition diagram

- What is a state transition diagram?
State transition diagram – example

FIGURE 7.4 State machine model of Stack without guards.
What are complete and incomplete state machine specifications?
Complete & incomplete specifications – 2

- Complete specifications
  - A transition for every event-state pair

- Incomplete specifications
  - The norm for modeling
    - For design too cumbersome to completely specify, as only a small subset is of interest

- Cannot ignore unspecified event-state pairs for testing
  - Why?
Equivalent states

- What are equivalent states?

- What problem exists with equivalent states?
Equivalent states

- Any two states are equivalent
  - If all possible event sequences applied to these states result in identical behaviour
  - By looking at the output cannot determine from which state machine was started
  - Can extend to any pair of states

- Minimal machine has no equivalent states
Equivalent states

- What problem exists with equivalent states?
Equivalent states

- A model with equivalent states is redundant
  - Probably incorrect
  - Probably incomplete
Reachability

- What is reachability?
Reachability – 2

- State $S_f$ is reachable from state $S_t$
  - If there is a legal event sequence that moves the machine from $S_f$ to $S_t$
  - Just stating a state is reachable implies reachable from the initial state
Reachability problems

- Using the notion of reachability, what problems does it show?
Reachability problems – 2

- Dead state
  - **Cannot leave**
    - Cannot reach a final state

- Dead loop
  - **Cannot leave**
    - Cannot reach a final state

- Magic state
  - **Cannot enter – no input transitions**
  - **Can go to other states**
    - Extra initial state
Guarded transitions

- What is a guarded transition?
Guarded transitions – 2

- The stack example state machine is ambiguous
  - There are two possible reactions to push and pop in the Loaded state

- Guards can be added to transitions
- A **guard** is a predicate associated with the event
- A **guarded transition** cannot fire unless the guard predicate evaluates to true
Guarded transitions – example

**Figure 7.5** State machine model of Stack with guards.
Limitations of the basic model

- Limited scalability
  - Even with the best tools available, diagrams with 20 states or more are unreadable

- Concurrency cannot be modeled
  - Different processes can be modeled with different state machines, but the interactions between them cannot

- Not specific enough for Object-Oriented systems
State transition diagram for traffic light.
Traffic light with superstates – all states view

Initial state

Superstates

Common to all inner states

FIGURE 7.12 Statechart for traffic light.
Traffic light – top level view

Traffic Light Controller
Traffic light – level 1 view

Traffic Light Controller

- PowerOn
- LiteOff
- Fault
- Reset(noFault)

States:
- Off
- On
Traffic light – level 2 view

Traffic Light Controller

- PowerOn
- Off
  - Reset(noFault)
  - Fault
  - LiteOff
- On
  - RedOn
  - Cycling
  - FlashingRed
  - FlashRedOn
Statechart advantages

- Easier to read
- Suited for object oriented systems (UML uses statecharts)
- Hierarchical structure helps with state explosion
- They can be used to model concurrent processes as well
Statechart problems

- Can vary in their details and implementation with different case systems
  - Need to be very careful when testing
Concurrent statechart

![Automotive control statechart with orthogonal states.](image)

**FIGURE 7.14** Automotive control statechart with orthogonal states.
State model

- Must support automatic test generation
- The following criteria must be met
  - Complete and accurate reflection of the implementation to be tested
  - Allows for abstraction of detail
  - Preserves detail that is essential for revealing faults
  - Represents all events and actions
  - Defines state so that the checking of resultant state can be automated
What is a state?

- We need an executable definition that can be evaluated automatically.
- An object with two Boolean fields has 4 possible states?
  - This would lead to trillions of states for typical classes.
Trillions of states

**Figure 7.16** Primitive view of a state space.
What is a state? – 2

- How can we address the problem?
What is a state? – 3

- A set of variable value combinations that share some property of interest
  - Can be coded as a Boolean expression
An example

- Consider the following class

```java
Class Account {
    AccountNumber number;
    Money balance;
    Date lastUpdate;
    ...
}
```

- The cross-product of all values is a primitive view of the state space
  - Yields too many states

- What abstraction gives fewer states?
- How is the abstraction represented?
Three abstract states

Shaded volumes
State invariants

- A valid state can be expressed with a state invariant
  - A Boolean expression that can be checked

- A state invariant defines a subset of the values allowed by the class invariant

  \texttt{ensure} \ a \ \texttt{or} \ b

  - In Eiffel this defines two possible states
Transitions

- A transition is a unique combination of
  - Two state invariants
    - One for the accepting
    - One for the resultant state
    - Both may be the same
  - An associated event
  - An optional guard expression
  - Optional action or actions
Transition events

- A message sent to the class under test
- A response received from a supplier of the class under test
- An interrupt or similar external control action that must be accepted
Transition actions & guards

- A guard
  - Predicate associated with an event
  - No side effects

- An action
  - The side effect that occurs
The initial state of an object is the state right after it is constructed.

However, a class may have multiple constructors that leave the object in different states.

To avoid modeling problems we define that an object is in the $\alpha$ state just before construction.

- $\alpha$ transitions go from $\alpha$ state to a constructor state.
Omega states

- Similarly with $\omega$ and destruction
  - *Not necessary to model* $\omega$ *for languages that have garbage collection*
  - $\omega$ *transitions go from a destructor state to the* $\omega$ *state*