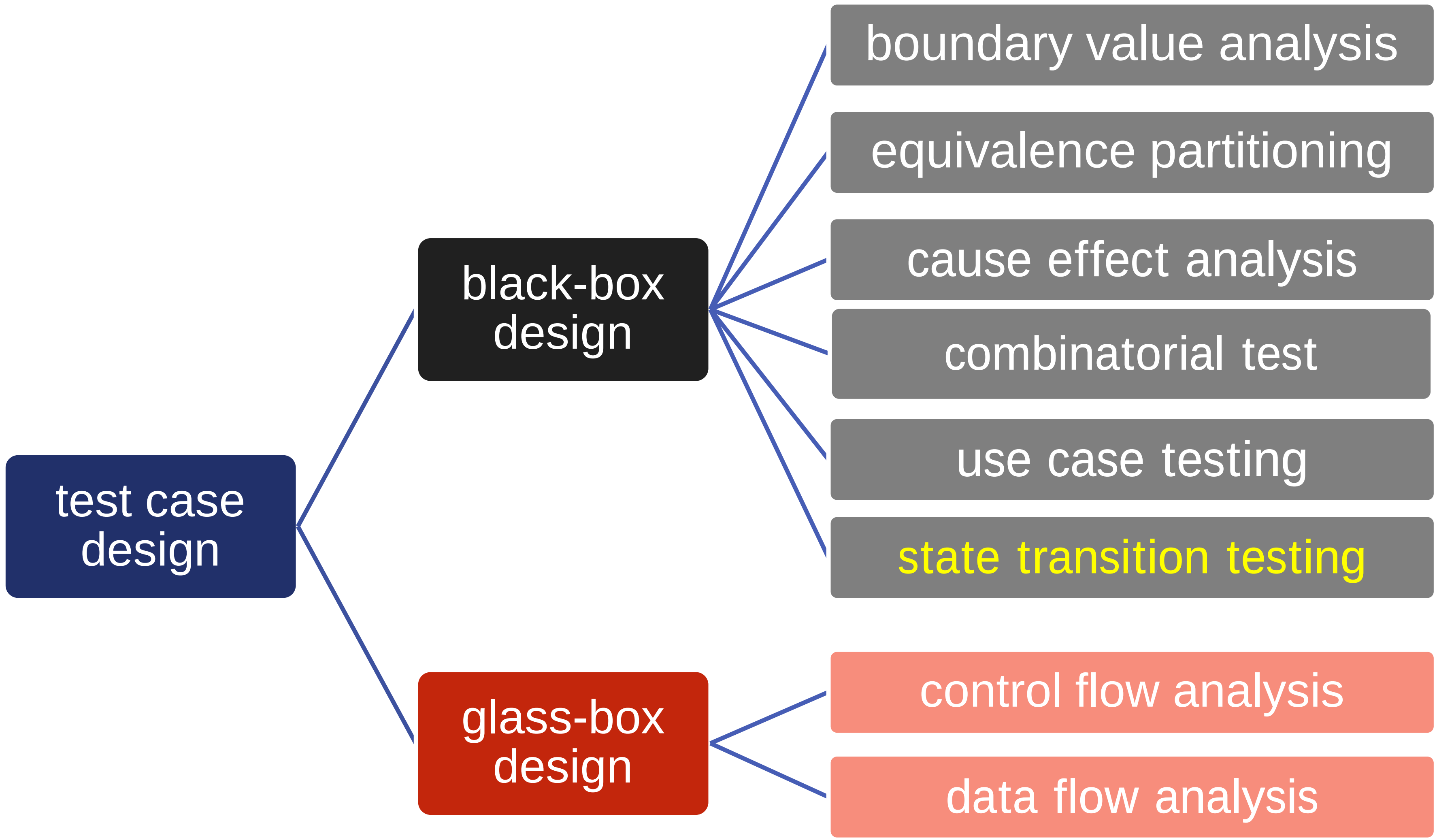
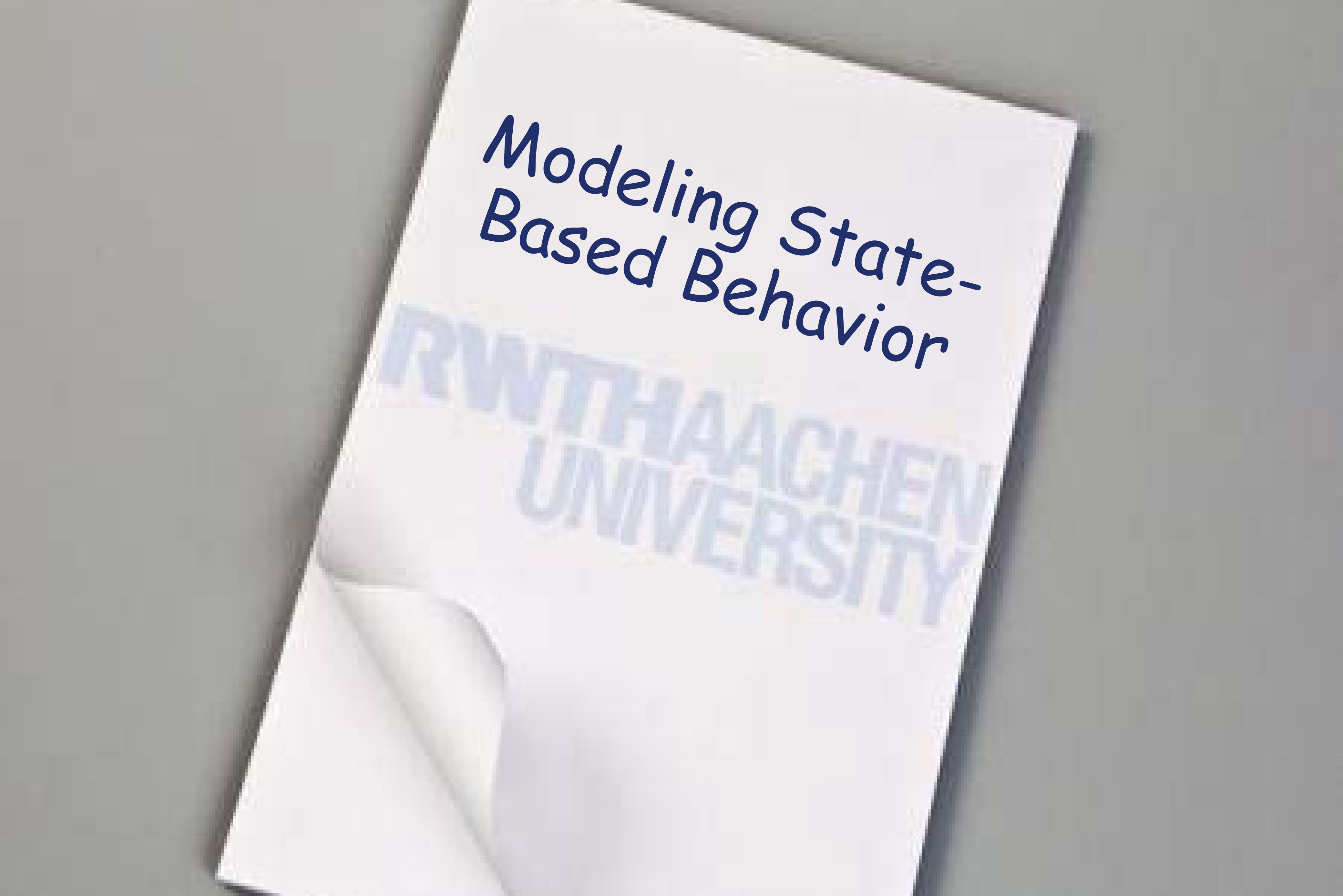
State Transition Testing

- Modeling State-Based Behavior
- STT Coverage
- N+ Testing Strategy
- JUnit Example

Approaches to Test Case Design



Modeling State-Based Behavior



State-based Behavior

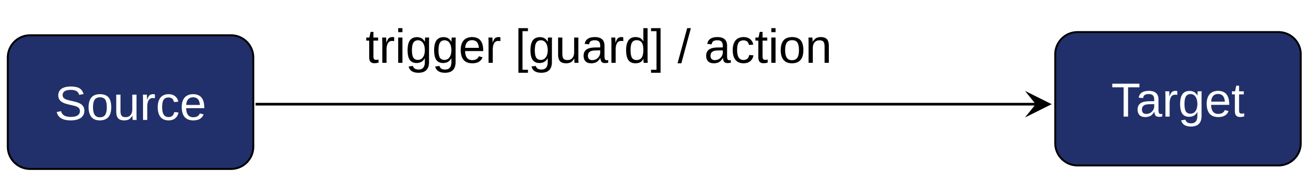
- Sometimes, systems or components implement state-based behavior
- Behavior can be specified using a state machine
  - Mature theory on state automata
  - Many different type of state automata
- UML State Diagrams
  - an object-based variant of Harel's state charts
  - have the characteristics of both Mealy and Moore machines

State Machine – States (Recap.)

- State machine
  - Event-ordered behavior specifying the sequences of states of a system
  - Events trigger transitions and cause responses
- State
  - Condition or situation in the life of a system during which it
    - satisfies some condition
    - performs some activity, or
    - waits for some events
- Activity
  - Ongoing non-atomic execution within a state
- Action
  - Specifies an atomic computation that gets done as an object makes a transition

Transition (Recap.)

- Relationship between two states
  - where the source state will enter the target state
  - when a specified event occurs and/or a specified condition is satisfied
- Optional: event trigger, guard condition, action
  - Automatic transition
    - occurs when the activity of the source state completes
  - Non-automatic transition
    - caused by a named event

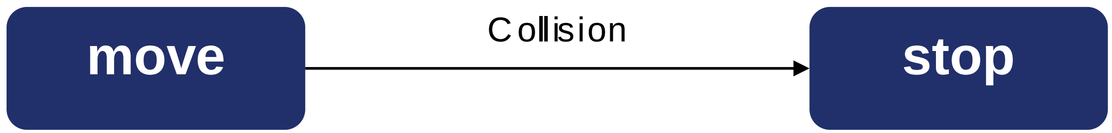


Events (Recap.)

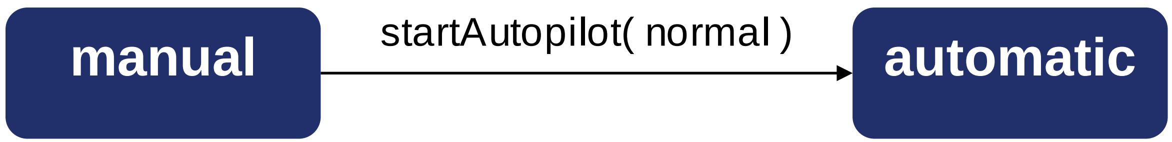
- Specification of a significant stimulus that has a location in time and space an can trigger a state transition.
- Signal event: `name`
  - explicit, named, asynchronous communication
- Call event: `operation`
  - explicit synchronous request among objects that wait for a response
- Change event: `when (boolean exp)`
  - change in value of a boolean expression
- Time event: `after (time)`
  - absolute time or a relative amount of time

Event Examples

Signal event



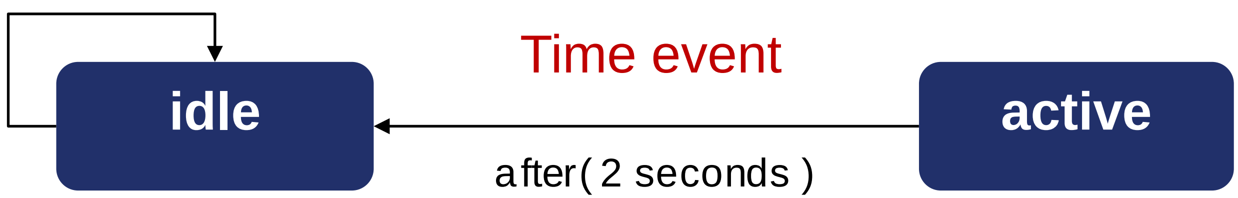
Call event



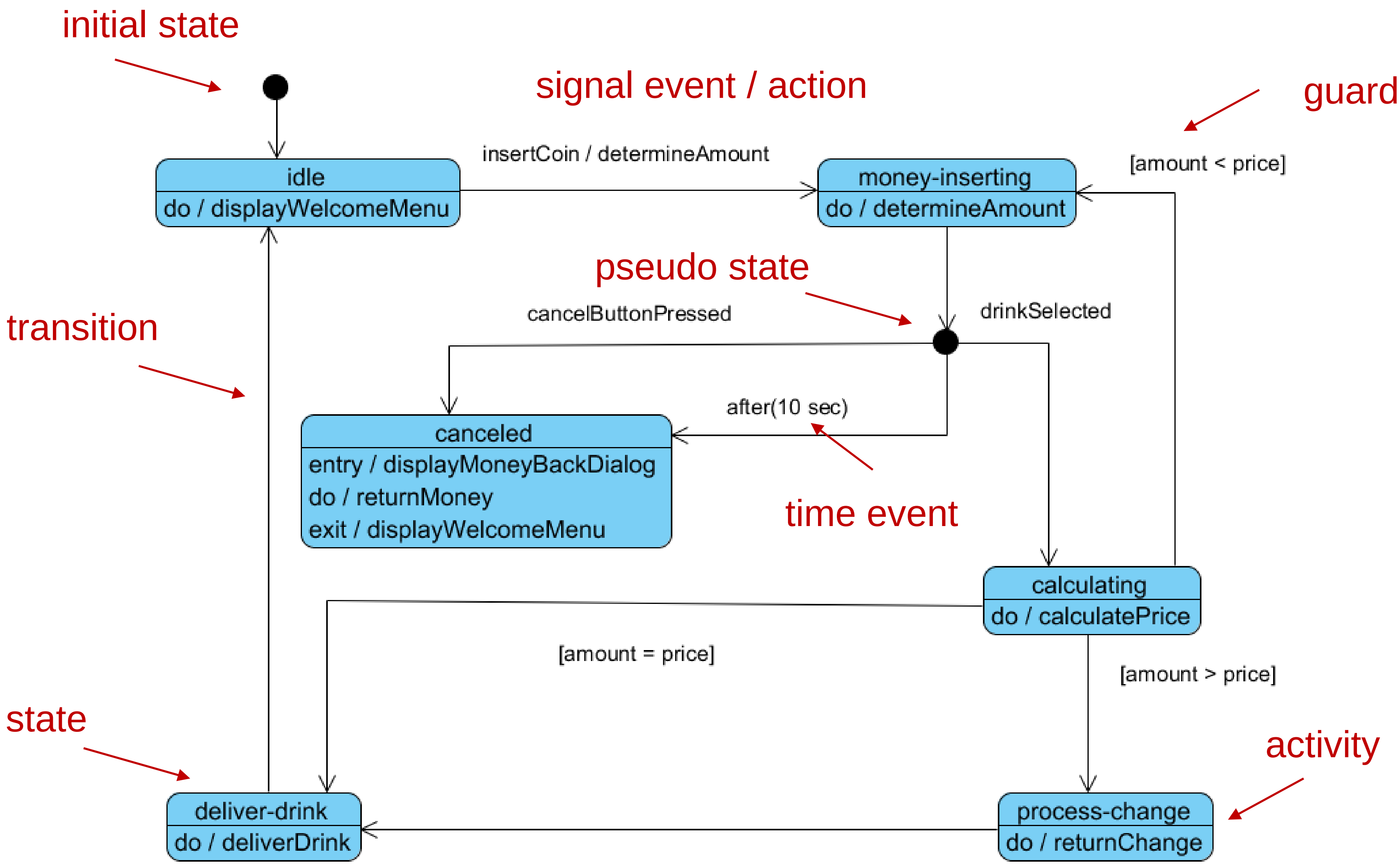
Change event

when(11:49PM)

Time event



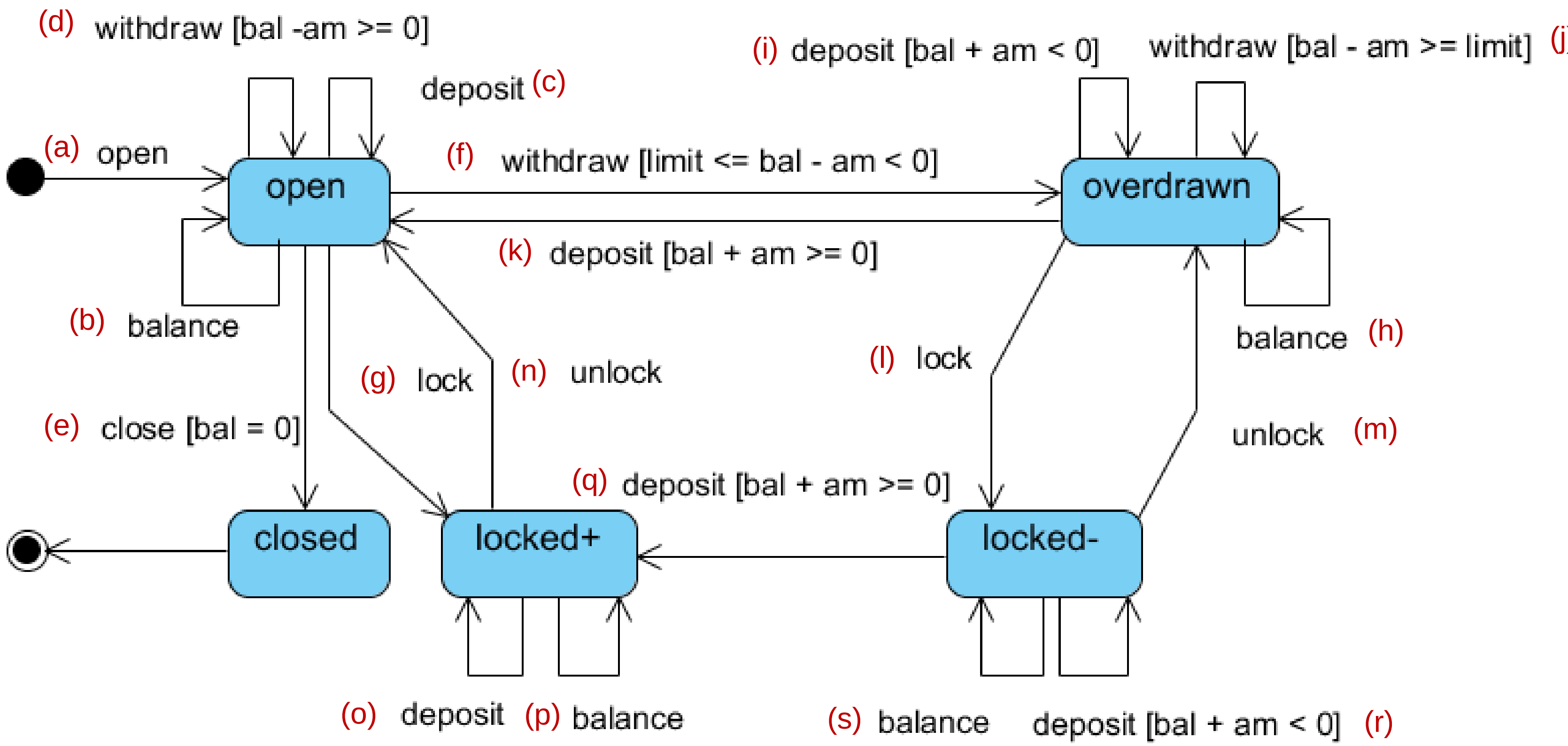
Example of UML State Diagram



Example: Simple Account - Requirements

1. Accounts have an overdraft facility
2. When created, the overdraft facility as well as an initial amount can be given
3. The holder can deposit money
4. The holder can withdraw money
5. Accounts can be locked and unlocked at any time by the bank
6. Accounts can be closed if the balance is zero

State Machine – Account



bal: balance
am: amount

STT Coverage

STT Error Model

- Missing or incorrect state
  - transition - resultant state is incorrect (but not corrupt)
  - event - valid message is ignored
  - action – wrong thing happens as a result of an transition
- Extra or corrupt state
  - unpredictable behavior

- Sneak path
  - message accepted when it should not
- Illegal message failure
  - unexpected message causes an error
- Backdoor (trap door)
  - implementation accepts undefined messages

State Transition Testing Coverage

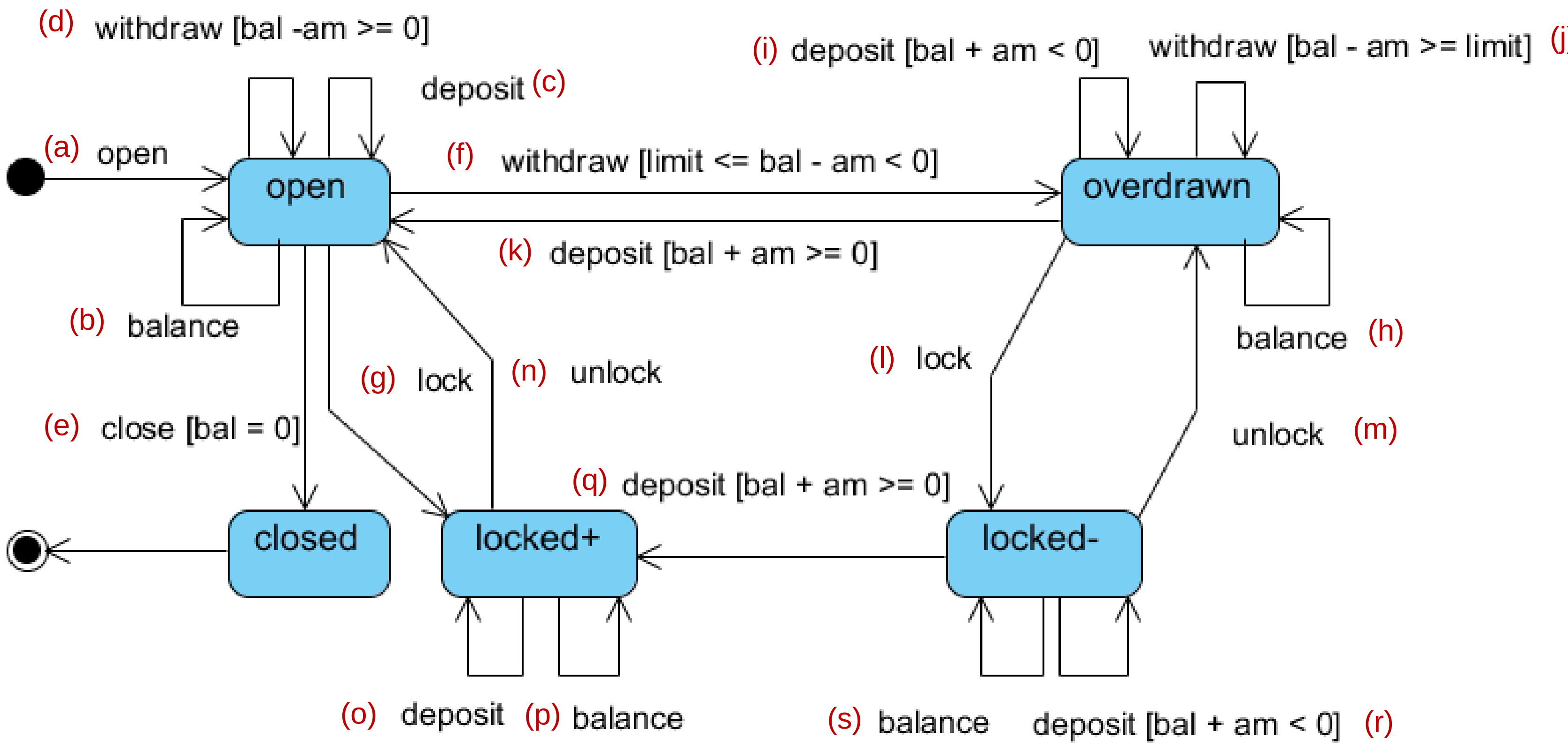
- piecewise
  - pieces of a certain kind are examined at least once
    - all-states coverage
    - all-events coverage
    - all-actions coverage
  - does not consider the structure of the state machine
  - only accidentally effective at finding errors

State Transition Testing Coverage

- all-transitions
  - every transition is examined at least once
  - no particular sequence required
  - minimum acceptable strategy

- all n-transition sequences
  - every transition sequence of n events is examined at least once
  - transition sequence is called switch
    - n-1 switch coverage
    - $n = 1 \rightarrow$ 0-switch coverage

State Machine – Account



bal: balance
am: amount

Chow's Switch Coverage

0 – switch

- Test every state transition
- → all actions, but no transition errors

1 – switch

- Test every sequence with two states transitions
- → all actions and some transition errors

2 – switch

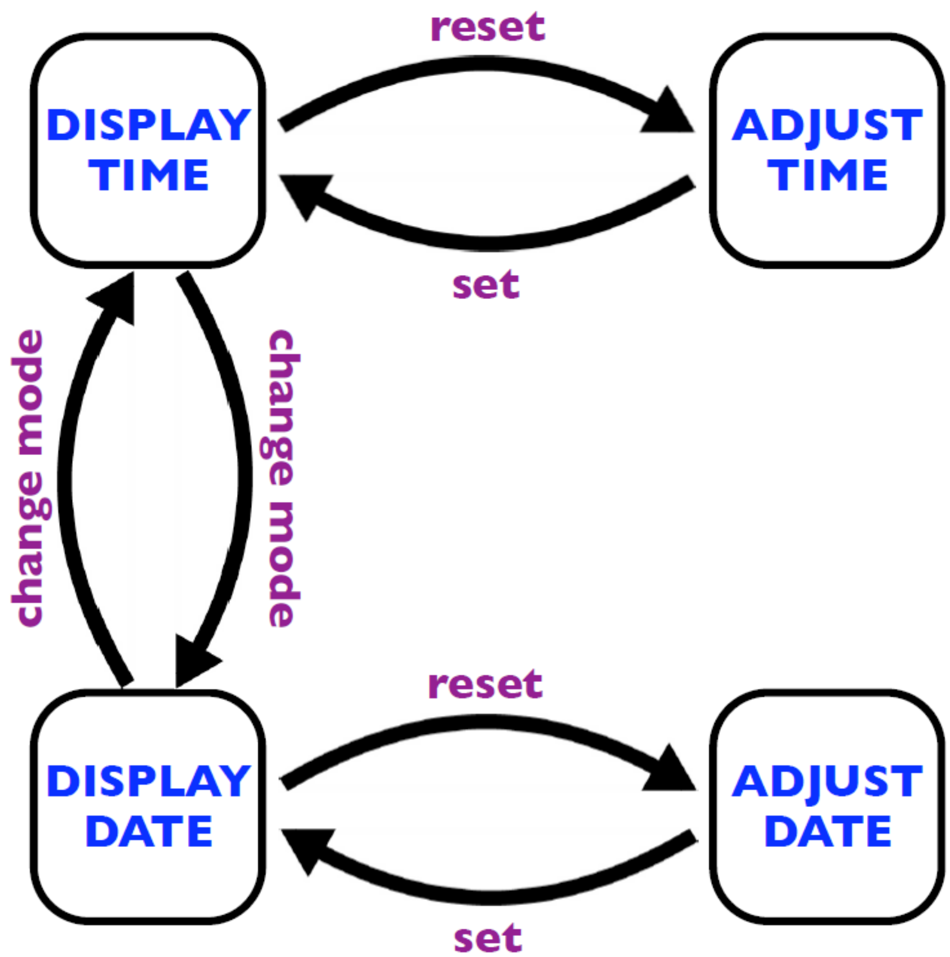
- Test every sequence with three state transitions
- ...

n-1-switch

- every sequence with n transitions
- → all actions/transition defects and additional states

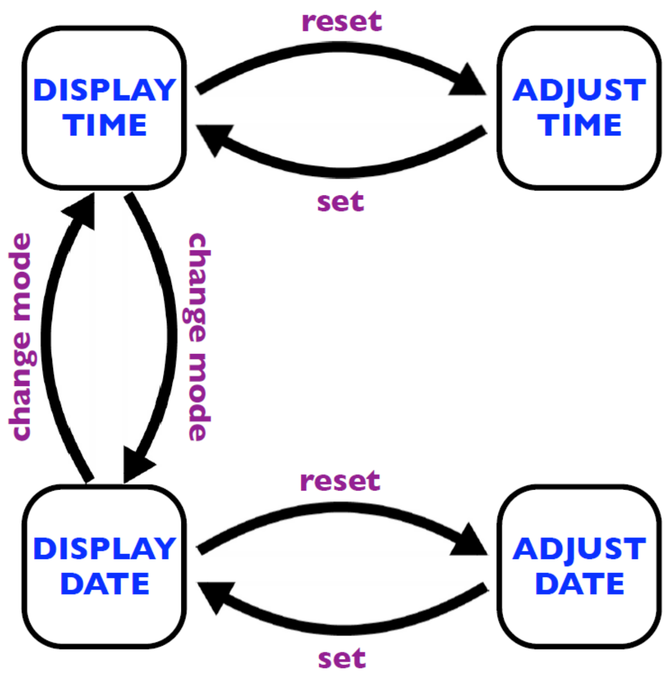
Tsun S. Chow, Testing Software Design Modeled by Finite-State Machines, IEEE Trans. on SE, Vol. 4, No. 3, Mai, 1978.

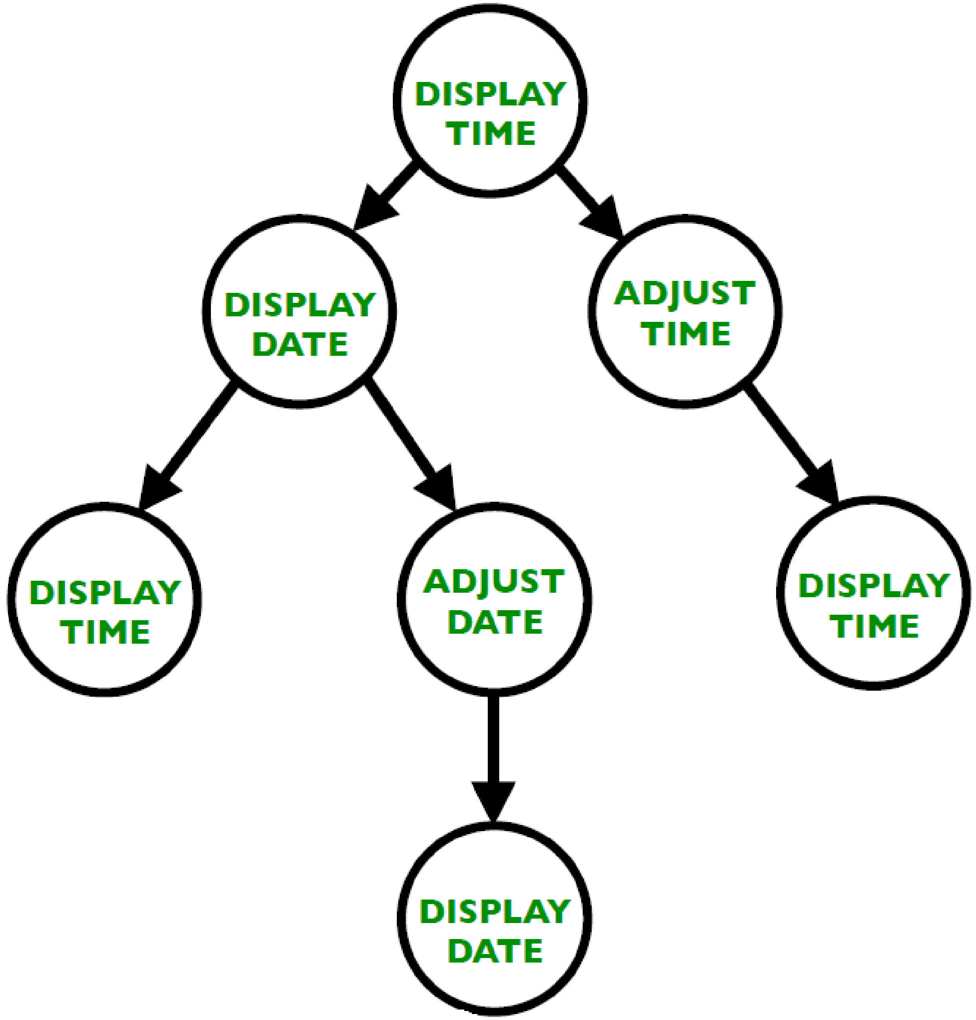
Example: Digital Clock



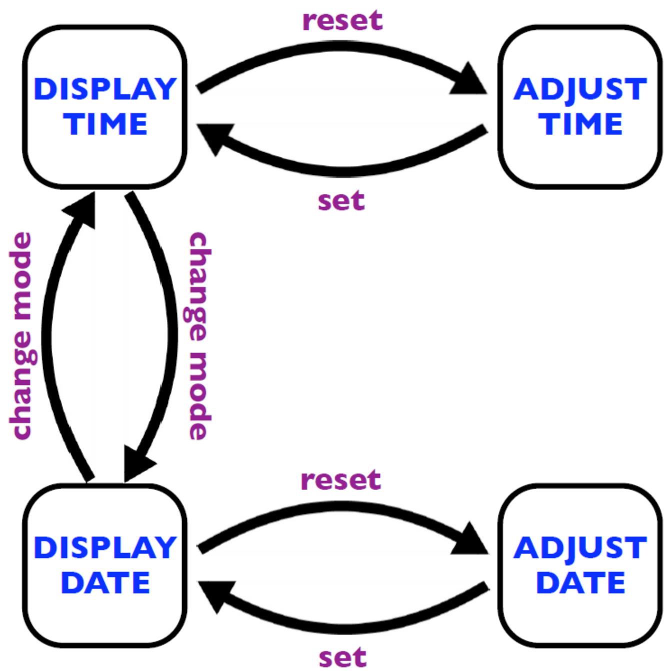
http://www.oxon.bcs.org/downloads/state_transition_testing_handout.pdf

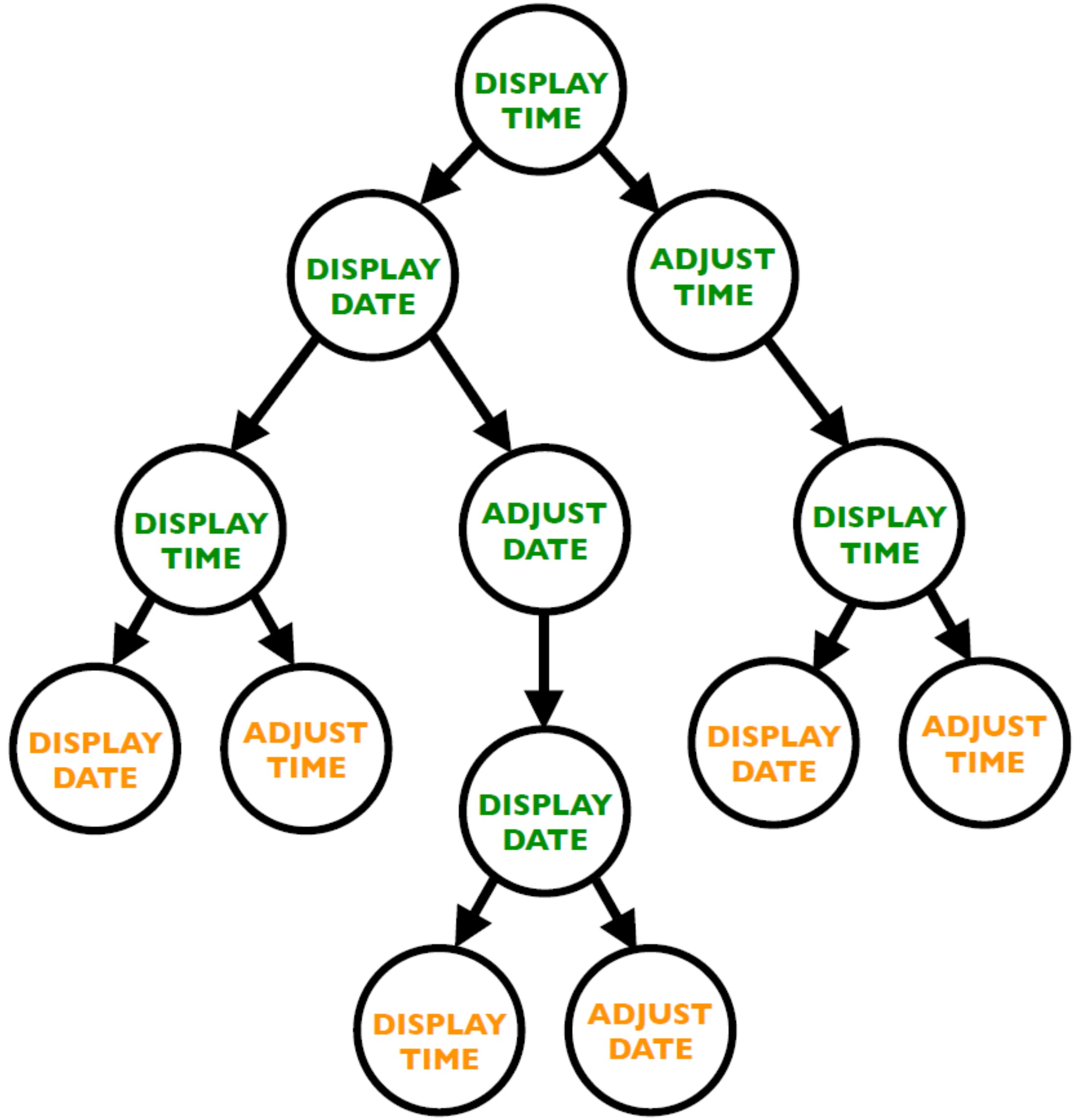
Example: 0-switch





Example: 1-switch

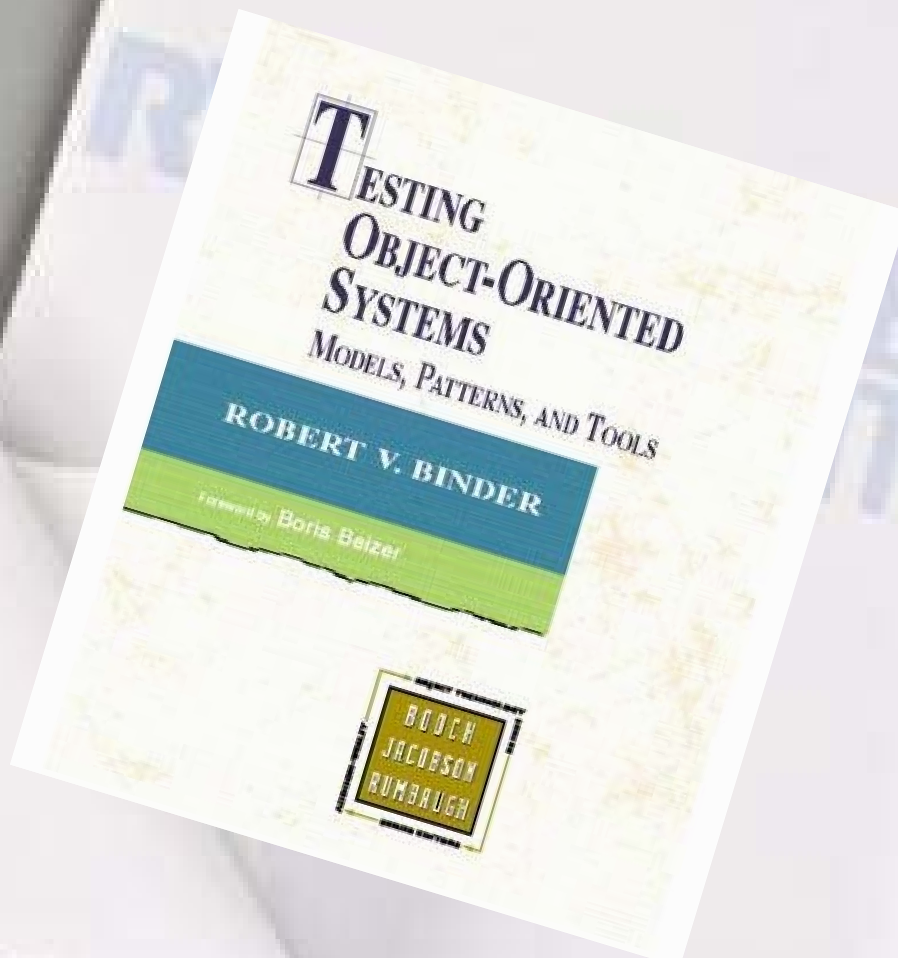




State Transition Testing Coverage

- all round-trip paths
  - every sequence of transitions beginning and ending in the same state is examined at least once
  - shortest round-trip path
    - transition to the same state
  - all simple path from start to final state
  - any sequence that goes beyond a round-trip must be part of a sequence belonging to another round-trip

N+ Test Strategy



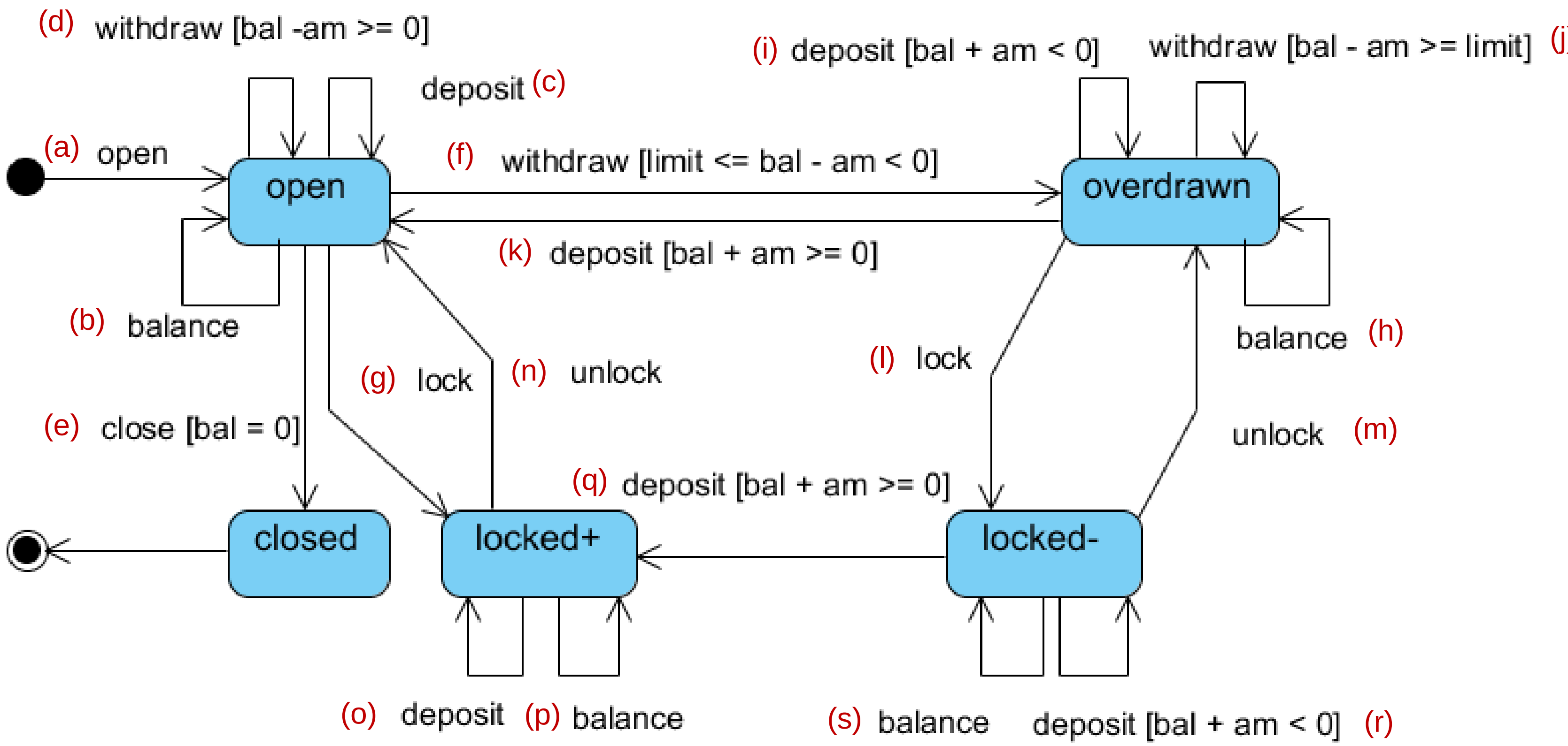
Modified N+ Test Strategy

1. Create a state model
2. Create the round-trip test cases
3. Create the sneak path test cases
4. Sensitize the transitions in each test case
5. Create test data for each test case

- Detects all state control errors, sneak paths and many corrupt states

Binder, R. V. (2000). Testing Object-Oriented Systems – Models, Patterns, and Tools. Addison-Wesley Longman, Reading, MA.

State Machine – Account



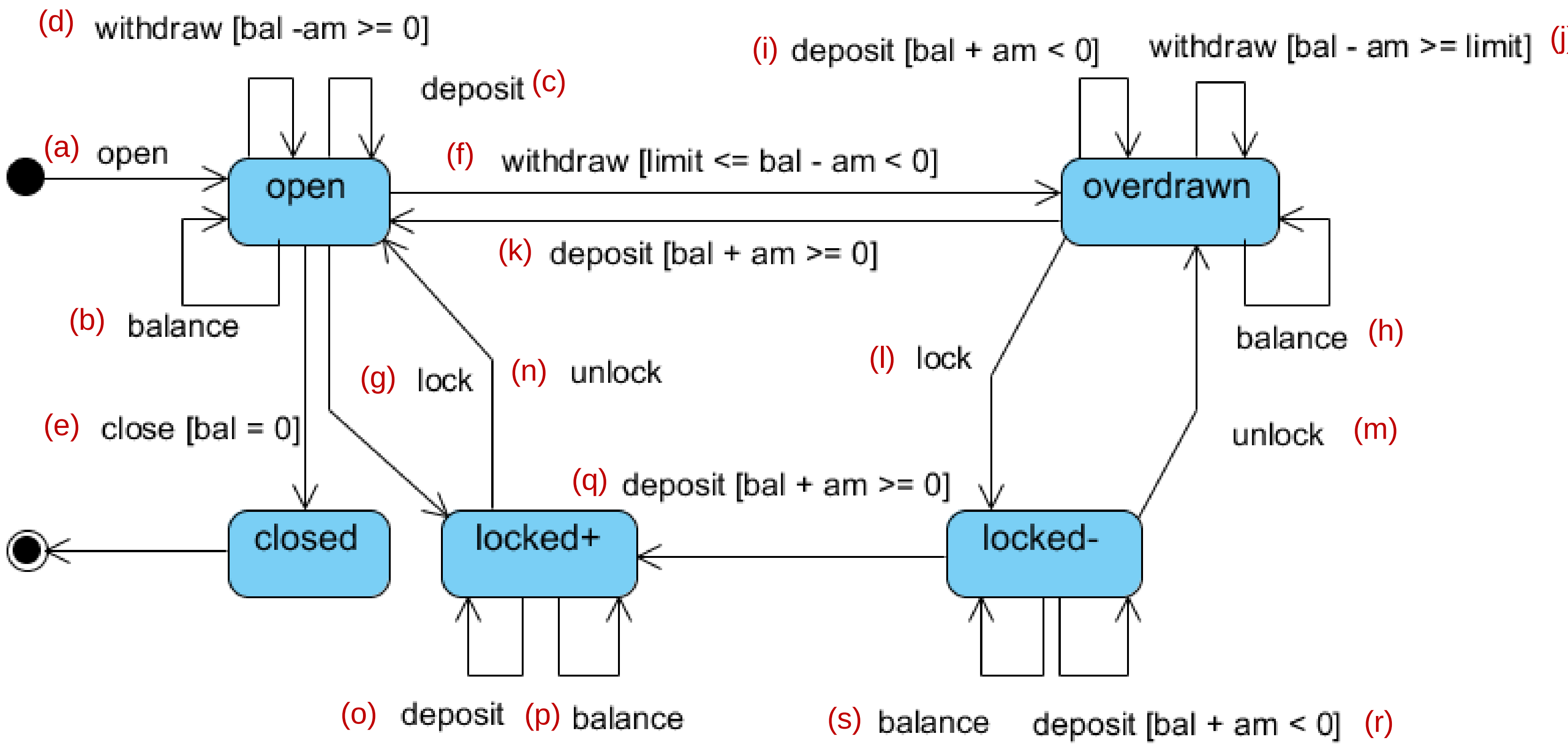
bal: balance
am: amount

State Transition Tree (Round-trip Path Tree)

1. The initial state is the root node of the tree (level 1)
2. Examine the state that corresponds to each non-terminal leaf node at level k:
   - Draw a new branch for every outgoing transition
   - Draw a new state at level k+1 in the state transition tree
   - Keep in mind guards (conditions)
   - Mark the new branch with a triple event/guard/action
   - Mark the new node with target state

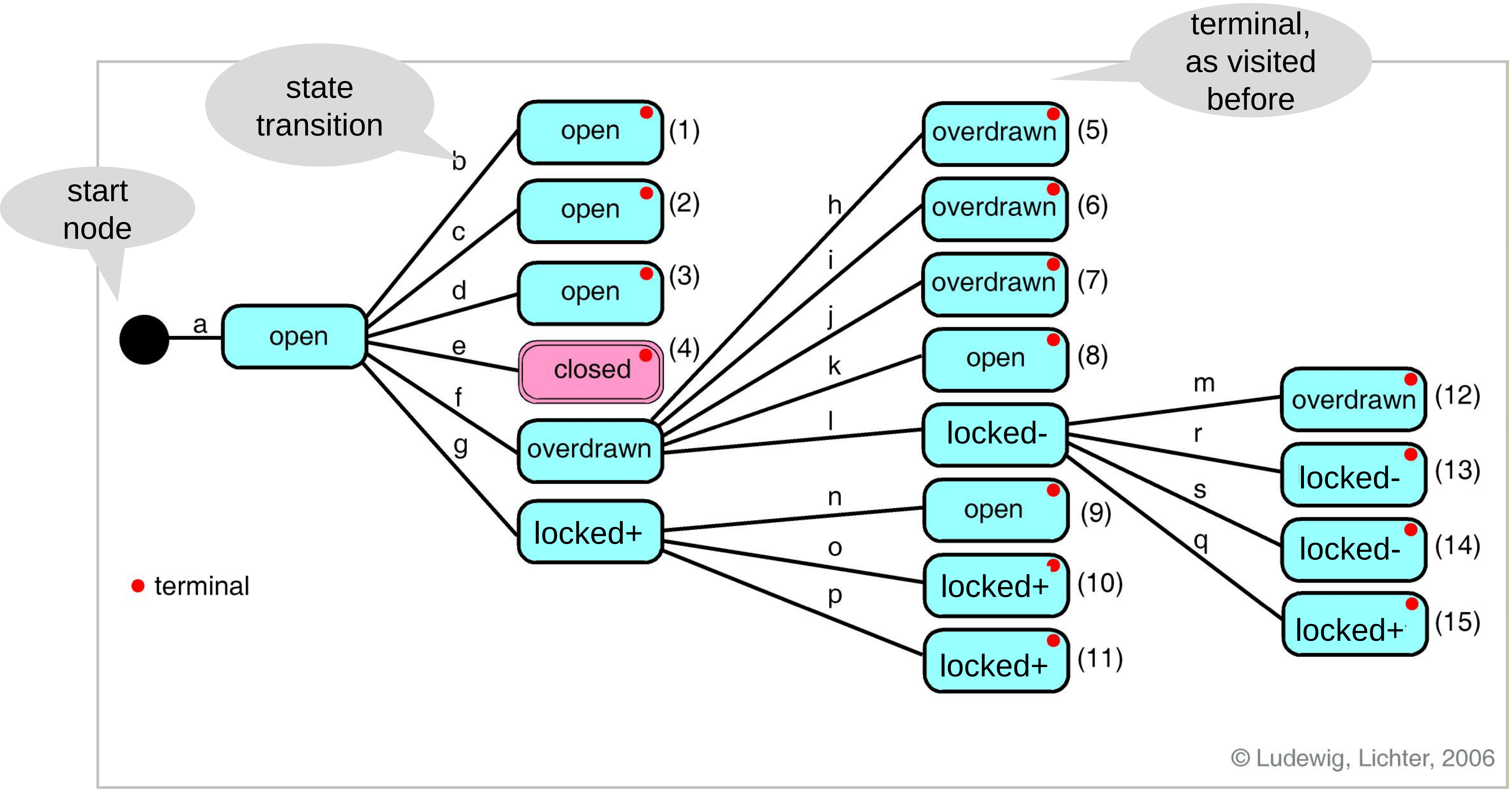
3. Mark every added node (state) from step 2 as terminal
   - if it is a finite state
   - if it already occurs somewhere in the state transition tree
4. Repeat step 2 and step 3 until all leaf nodes are marked terminal

State Machine – Account



bal: balance
am: amount

Example – Account



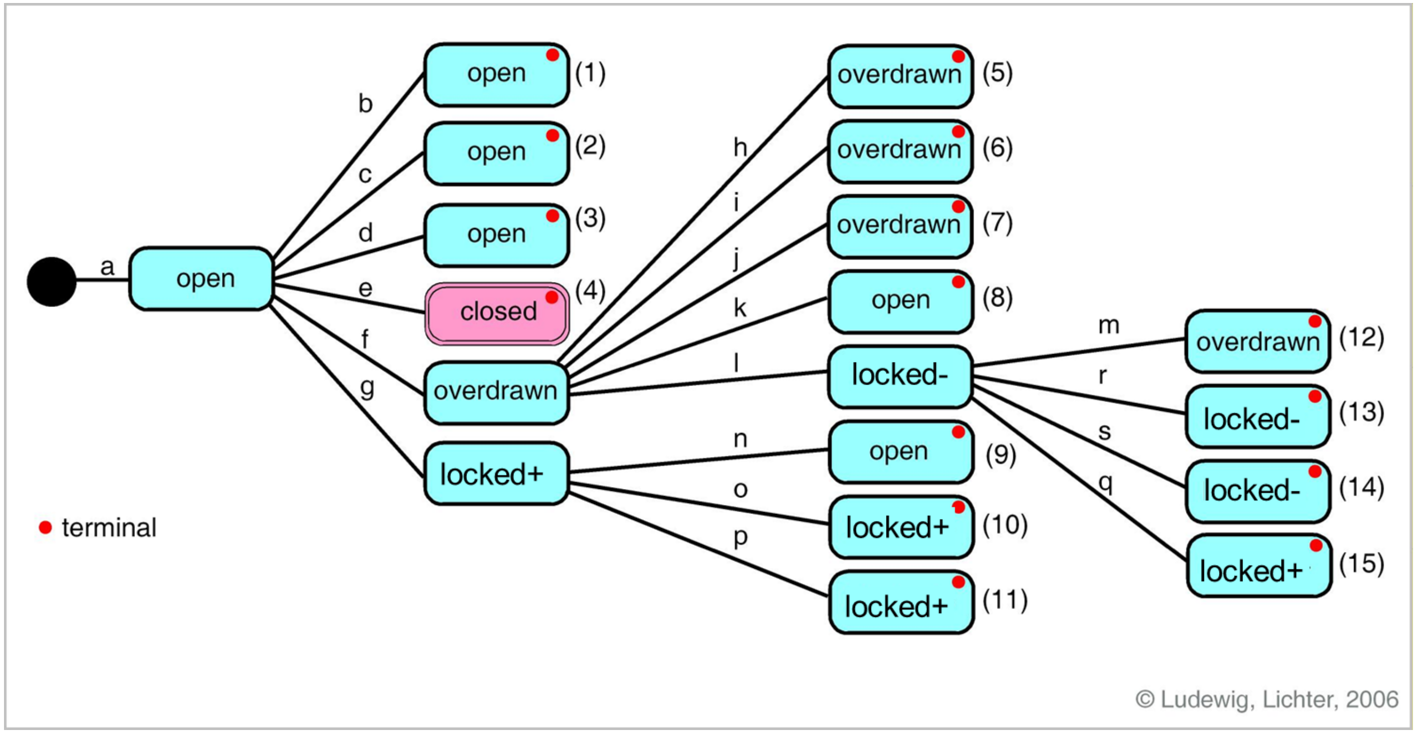
State Transition Tree - Remarks

- Creates all round-trip path
  - Round-trip path starts in root node and ends
    - in a finite state
    - in the first already walked-through node
  - i.e. in case of loops: the round trip path contains exactly one iteration
- The tree structure depends on the order in which transitions are traced
  - A depth first search yield fewer, longer test sequences
  - The order in which states are investigated is supposed to be irrelevant

Round Trip Test Cases

- Derived from state transition tree
  - Path from root node to a leaf
- Test conformance of implementation and state machine
  - Detect all action errors and all state transition errors
  - Detects missing states and some corrupt states

Example – Account



trip	start state, events, intermediate states	final state, bal
7	OPEN, withdraw (100) → OVERD, withdraw (200)	OVERD, -300
8	OPEN, withdraw (100) → OVERD, deposit (500)	OPEN, 400
15	OPEN, withdraw (100) → OVERD, block → LOCKED-, deposit (300)	LOCKED+, 200

Discovering Sneak Paths

- If state machines are incompletely specified, we have to test for sneak paths
- Sneak path
  - are unexpected transitions
  - possible for each unspecified transition
  - possible for guarded transitions that evaluate to false
- Test of all state's illegal events
  - check that appropriate action is taken, e.g. exception handling
  - no need to check for sneak paths traversing two or more states

Particularly important for safety-critical systems!

Sneak Path Testing Procedure

1. Place the SUT in the required state
   - e.g. via round trip test case
2. Apply illegal event
   - by sending message or
   - forcing the test environment to generate the desired event
3. Check that the actual response matches the specified response
   - raise exception
   - create error message
4. Check that the resultant state is unchanged

Event-Response-Matrix

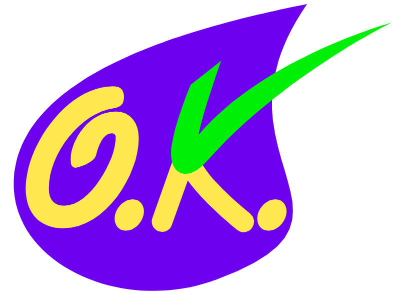
Event / Guard	OPEN	OVERD	LOCKED+	LOCKED-
balance	+	+	+	+
block	+	+	x	x
unblock	x	x	+	+
close [bal = 0]	+	-	x	-
close [bal <> 0]	x	x	x	x
deposit [bal + am >=0]	+	+	+	+
deposit [bal + am <0]	-	+	-	+
withdraw [bal - am >=0]	+	-	x	-
withdraw [limit <= bal-am <0]	+	+	x	x
withdraw [bal-am < limit]	x	x	x	x

error

impossible!

Inspection of States

- Inspection of states
  - Has to be done after every event
  - Prerequisite for N+ strategy



- SUT needs to offer state reporters
  - Services to access state information
  - Optionally special access services are needed for state invariants:

```
boolean isOpen();
boolean isOverdrawn();
```

JUnit Example

Class Account

```
public class Account {

  private enum State {OPEN, CLOSED, OVERDRAWN, LOCKED_N, LOCKED_P};
  private double balance;
  private double limit;
  private State state;

  public Account(double lim, double start)

  public double balance()
  public void deposit(double amount)
  public double withdraw(double amount)
  public void lock()
  public void unlock()
  public void close()

  public boolean isOpen()
  public boolean isClosed()
  public boolean isOverdrawn()
  public boolean isLocked_N()
  public boolean isLocked_P()
```

States

State reporter

Junit Conformance Tests

trip	start state, events, intermediate states	final state, bal
7	OPEN, withdraw (100) → OVERD, withdraw (200)	OVERD, -300
15	OPEN, withdraw (100) → OVERD, block → LOCKED-, deposit (300)	LOCKED+, 200

```
@Test
public void rt7() throws OperationNotAllowed {
  assertEquals(true, ac1.isOpen());
  ac1.withdraw(100);
  assertEquals(true, ac1.isOverdrawn());
  ac1.withdraw(200);
  assertEquals(-300, ac1.balance());
  assertEquals(true, ac1.isOverdrawn());
}
```

```
@Before
public void setAccount() throws Exception {
  ac1 = new Account(1000.0, 0);
}
```

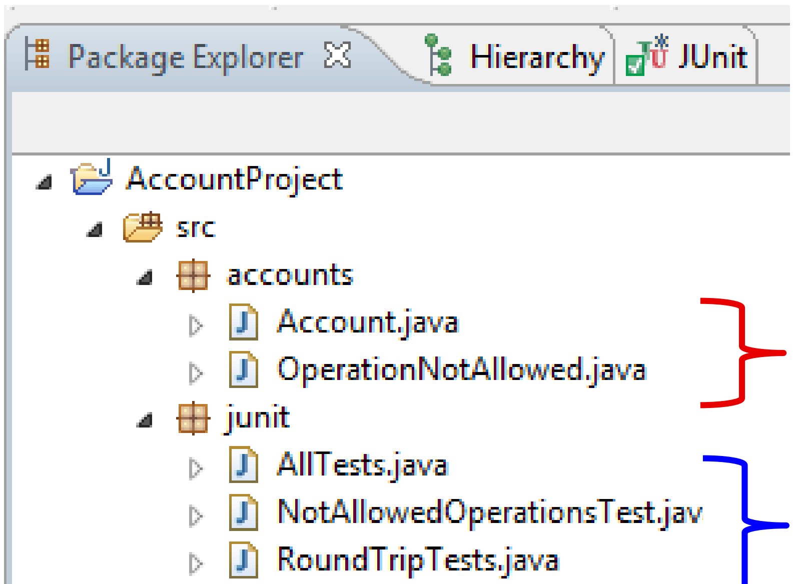
```
@Test
public void rt15() throws OperationNotAllowed {
  assertEquals(true, ac1.isOpen());
  ac1.withdraw(100);
  assertEquals(true, ac1.isOverdrawn());
  ac1.lock();
  assertEquals(true, ac1.isLocked_N());
  ac1.deposit(300);
  assertEquals(200, ac1.balance());
  assertEquals(true, ac1.isLocked_P());
}
```

Junit Sneak Path Tests

Event / Guard	OPEN	OVERD	LOCKED+	LOCKED-
close [bal = 0]	+	-	x	-
close [bal <> 0]	x	x	x	x

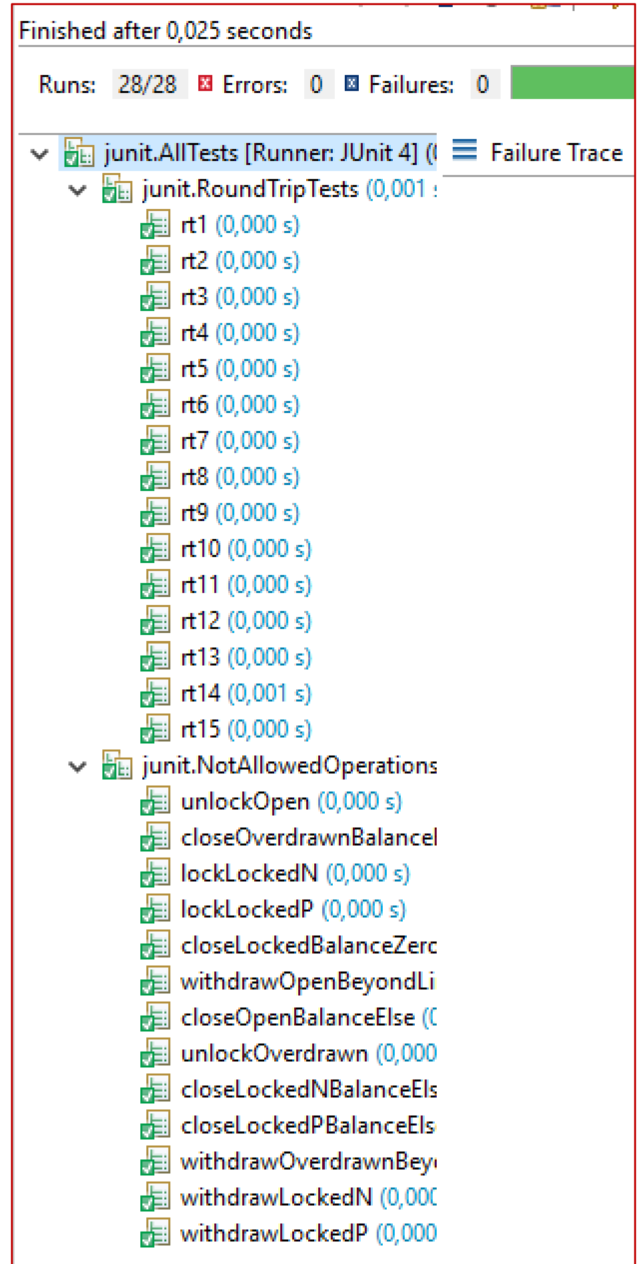
```
@Test (expected = OperationNotAllowed.class)
public void closeLockedPbalanceZero () throws OperationNotAllowed{
  ac1.lock();
  assertEquals(true, ac1.isLocked_P());
  assertEquals(0, ac1.balance());
  ac1.close();
}

@Test (expected = OperationNotAllowed.class)
public void closeOpenBalanceElse () throws OperationNotAllowed{
  ac1.deposit(100);
  assertEquals(true, ac1.isOpen());
  assertEquals(true, ac1.balance() != 0);
  ac1.close();
}
```

Some Remarks



126 LOC

228 LOC



Discussion – Summary

States n	Events k	Test Cases k * n	Messages k^2 * n/2
7	9	63	284
15	30	450	6750
30	100	3000	150.000

Assumption:
k/2 messages per test case

- Effective method
  - very little experience exists with these coverage criteria
- Effort
  - Depends heavily on size of state machine