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

Question 1

- **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**

Building blocks of a state machine

- **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

State machine behaviour

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 the **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

- Algorithms for output creation are not part of the model
- The firing of a transition does not consume any amount of time
  - **An event with no beginning or ending, which implies duration**

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

Question 2

- **What is a state transition diagram?**

State transition diagrams

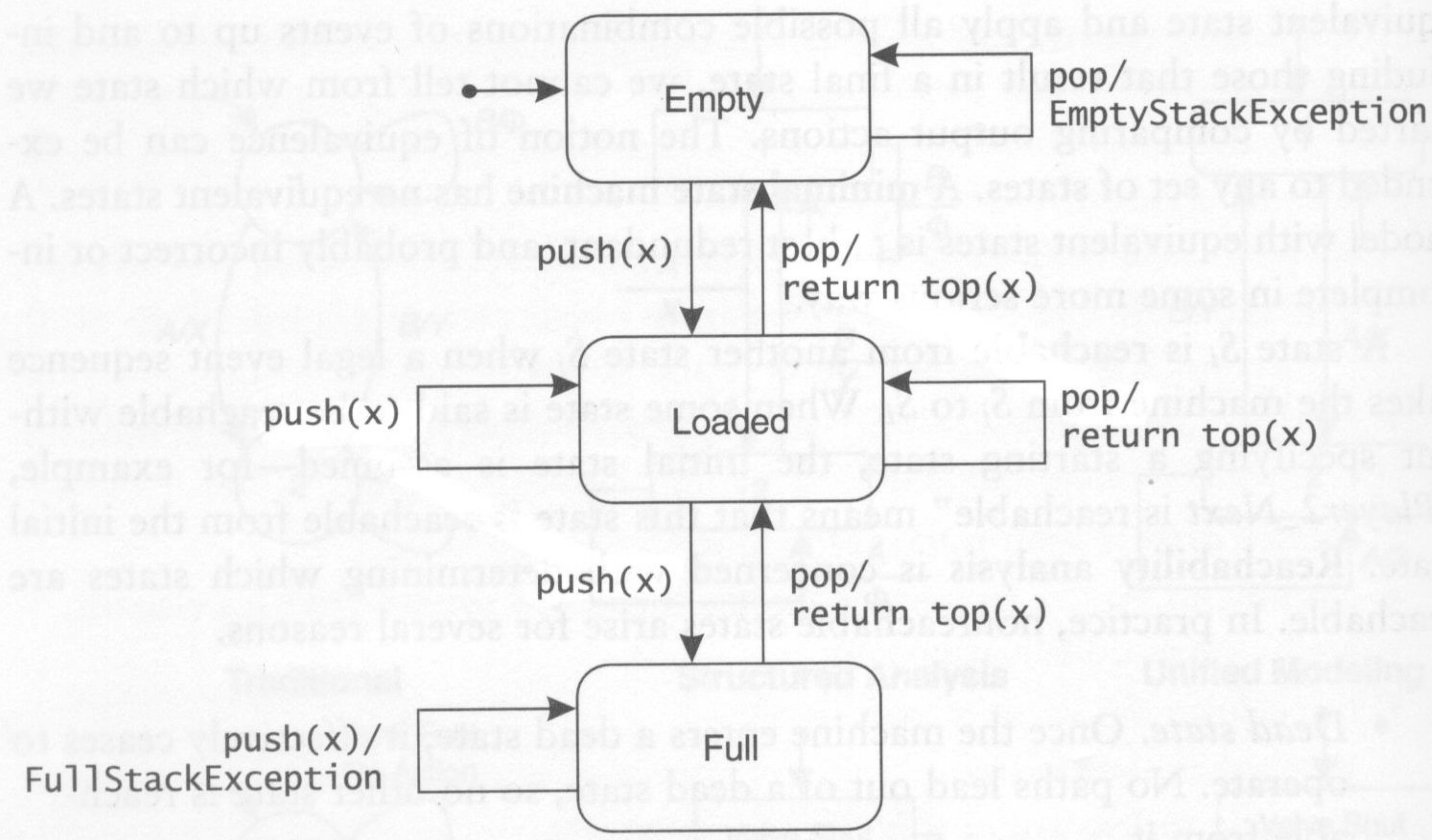


FIGURE 7.4 State machine model of Stack without guards.

Guarded transitions

- The previous model is ambiguous, e.g. 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

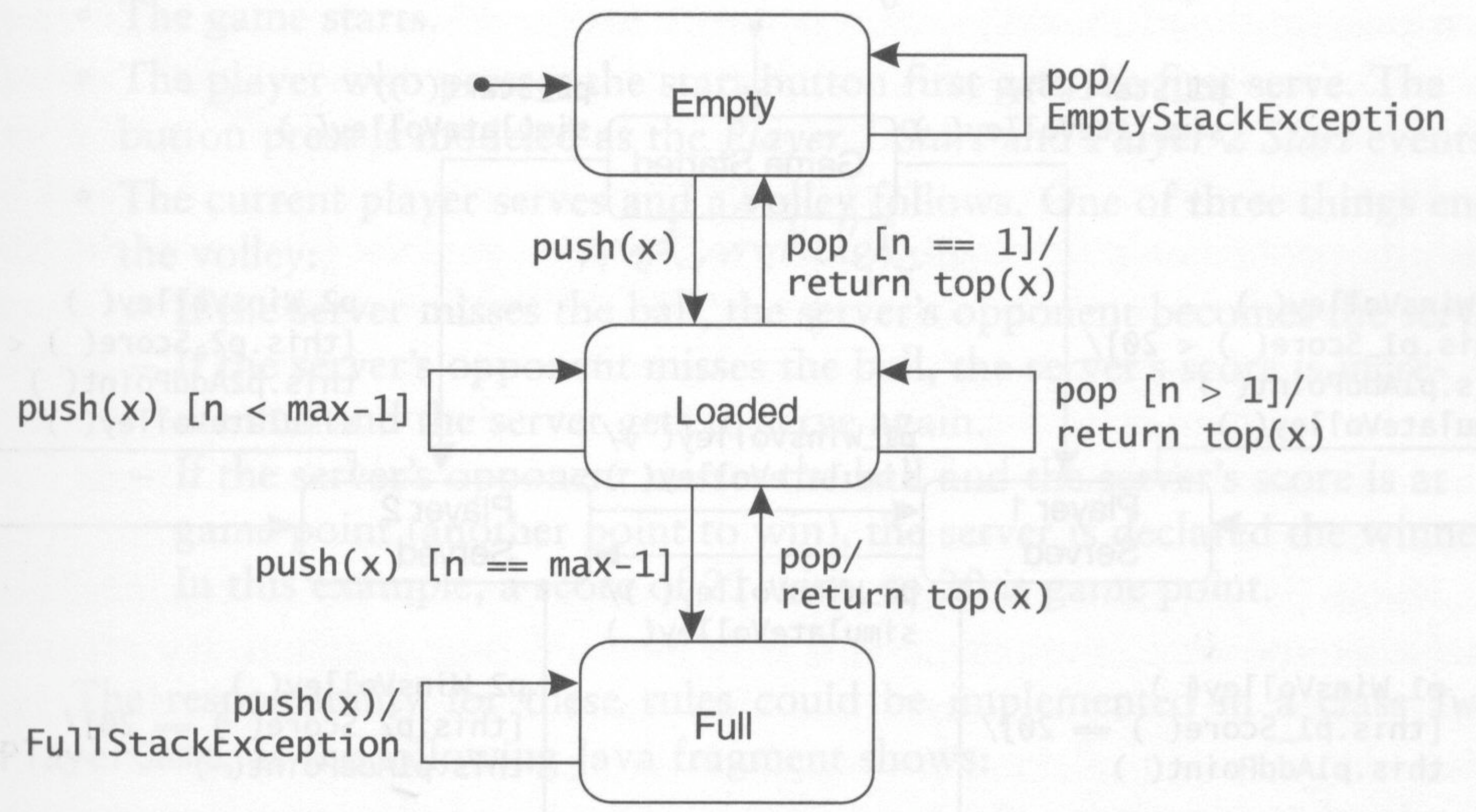


FIGURE 7.5 State machine model of Stack with guards.

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

Scalability – traffic light example

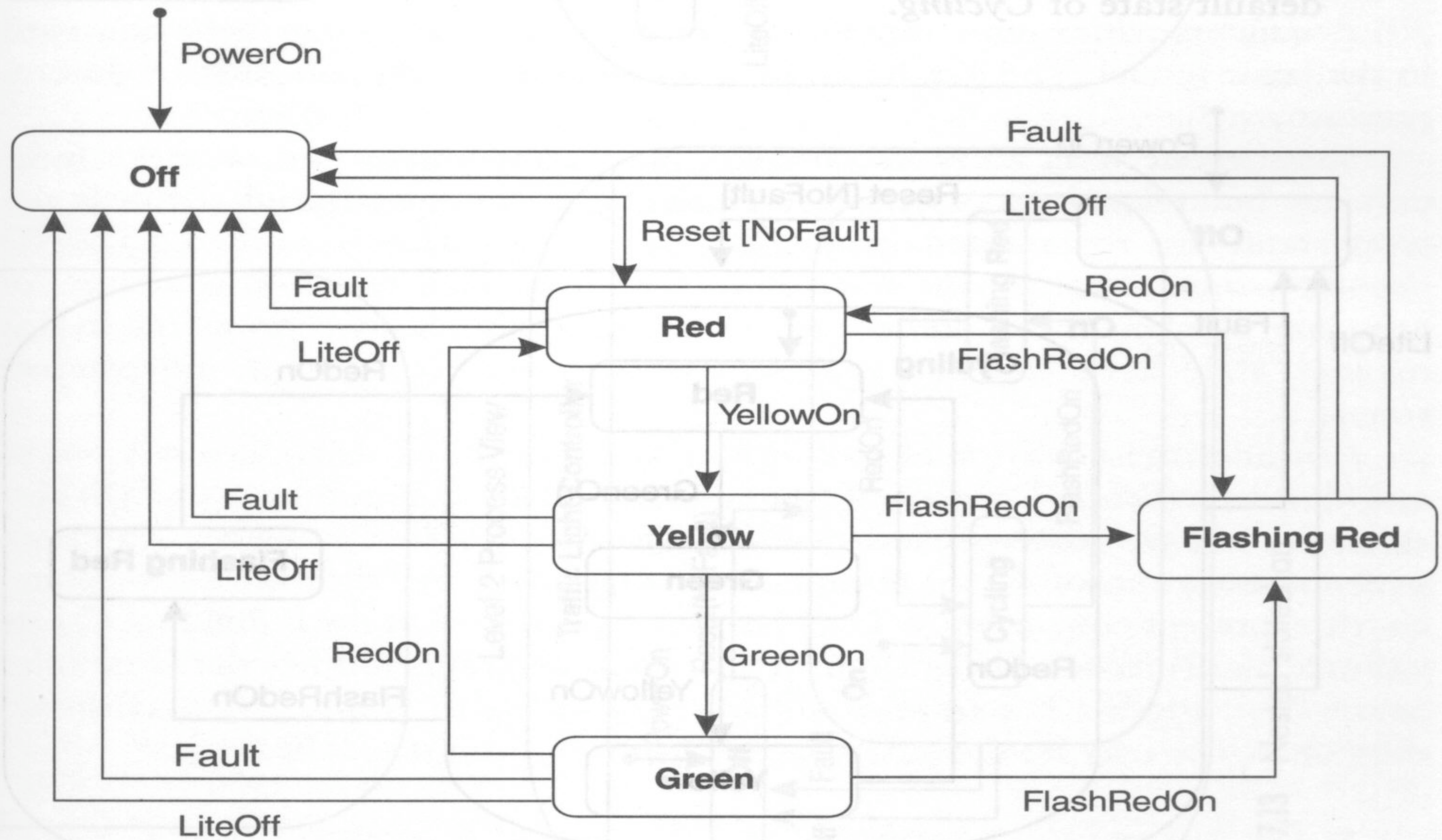


FIGURE 7.11 State transition diagram for traffic light.

Traffic light with superstates – all states view

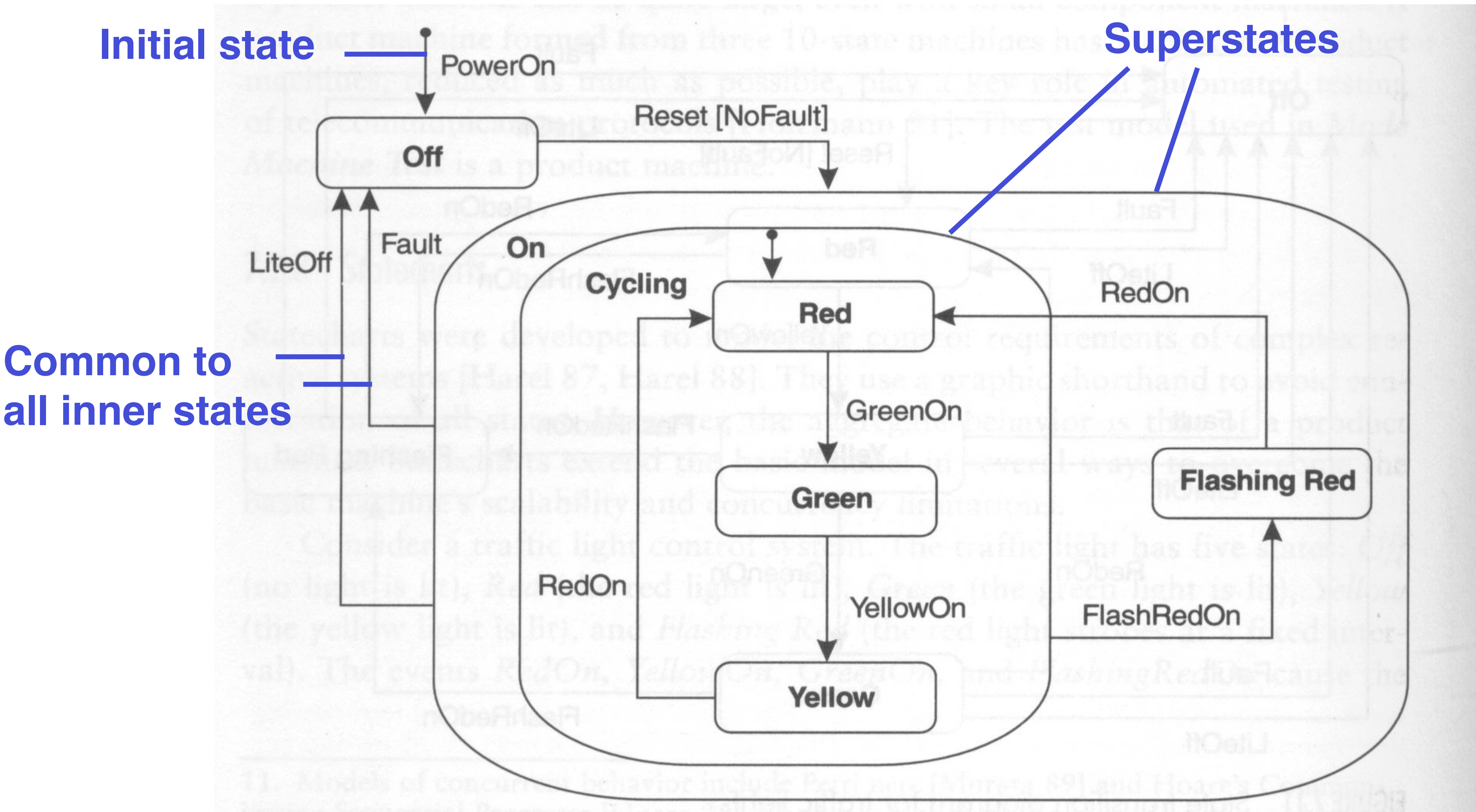
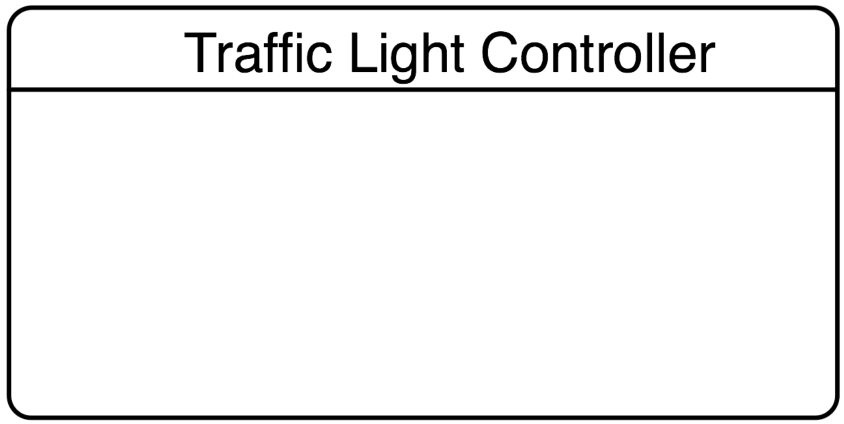
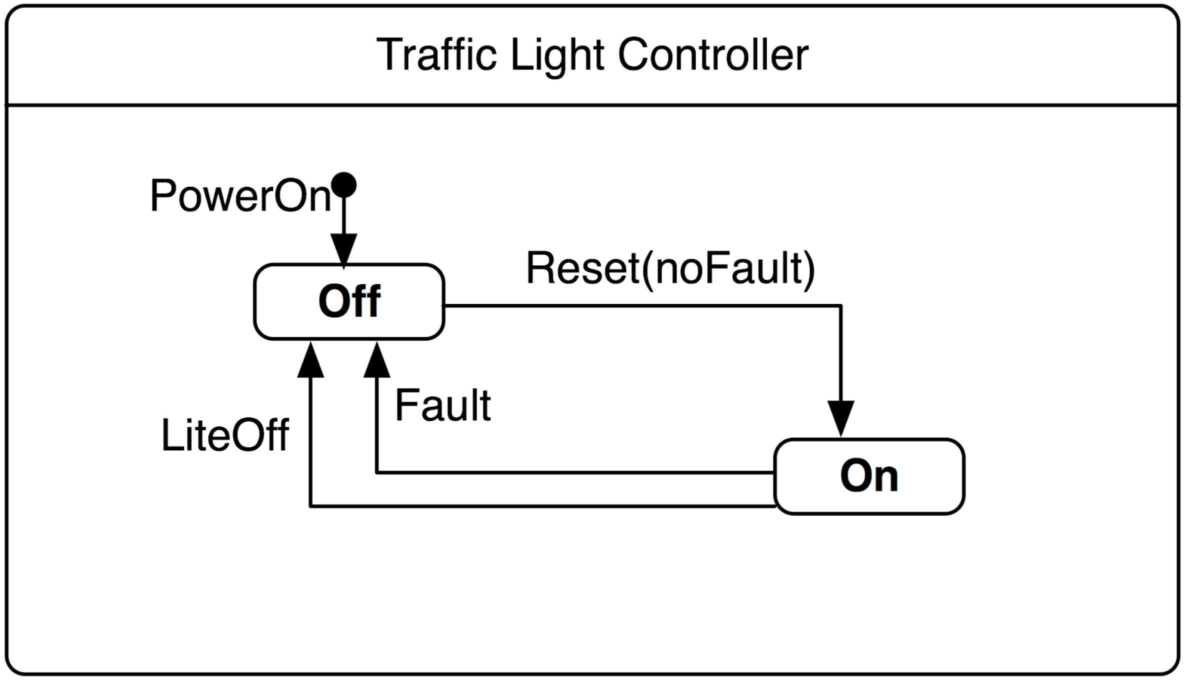


FIGURE 7.12 Statechart for traffic light.

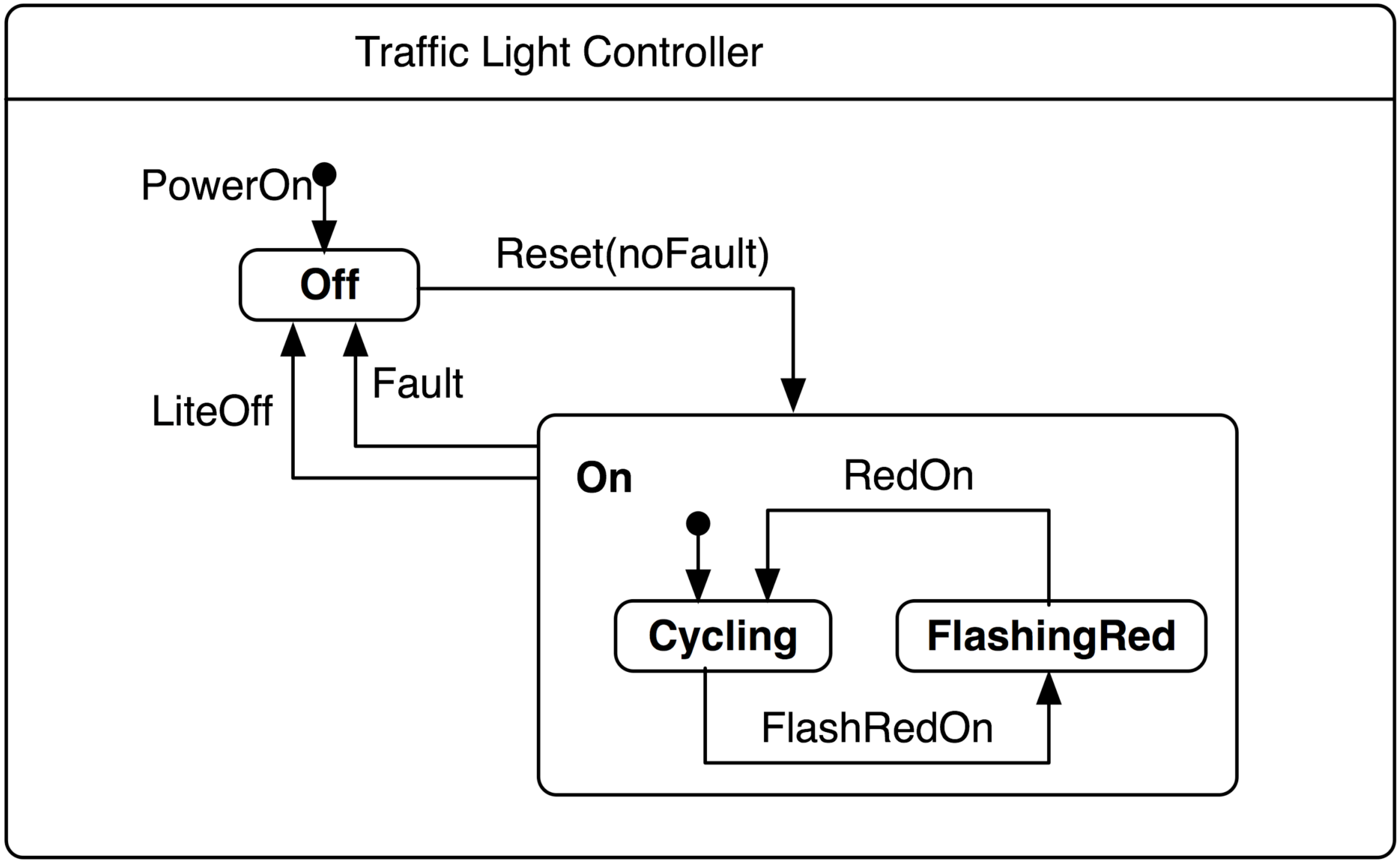
Traffic light – top level view



Traffic light – level 1 view



Traffic light – level 2 view



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

Concurrent statechart

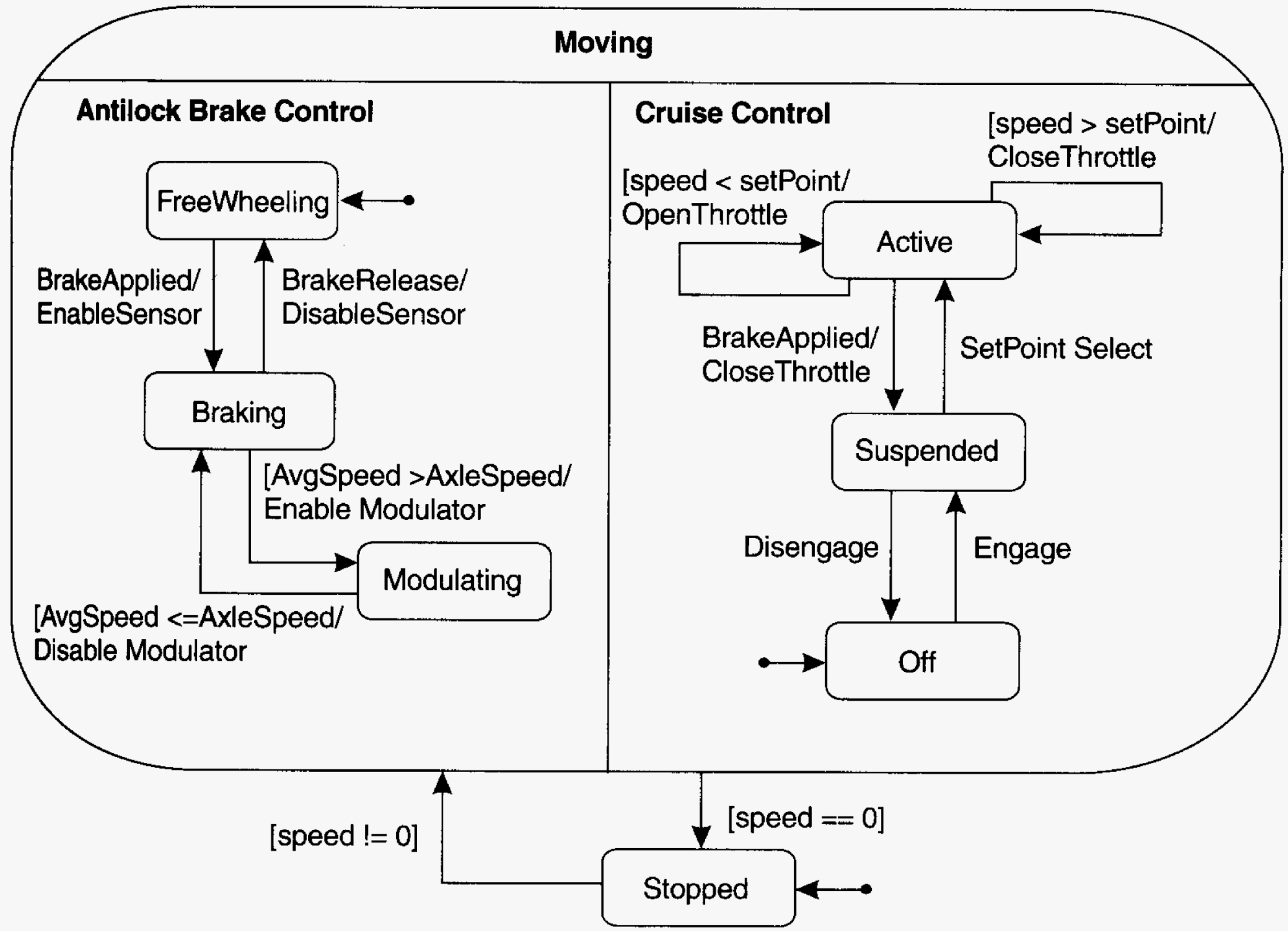


FIGURE 7.14 Automotive control statechart with orthogonal states.

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**
- Instead, state is
  - **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

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

- A primitive view of the state space would yield too many states
  - **The cross-product of all values**
- **What abstract gives fewer states?**
- **How is the abstraction represented?**

Trillions of states

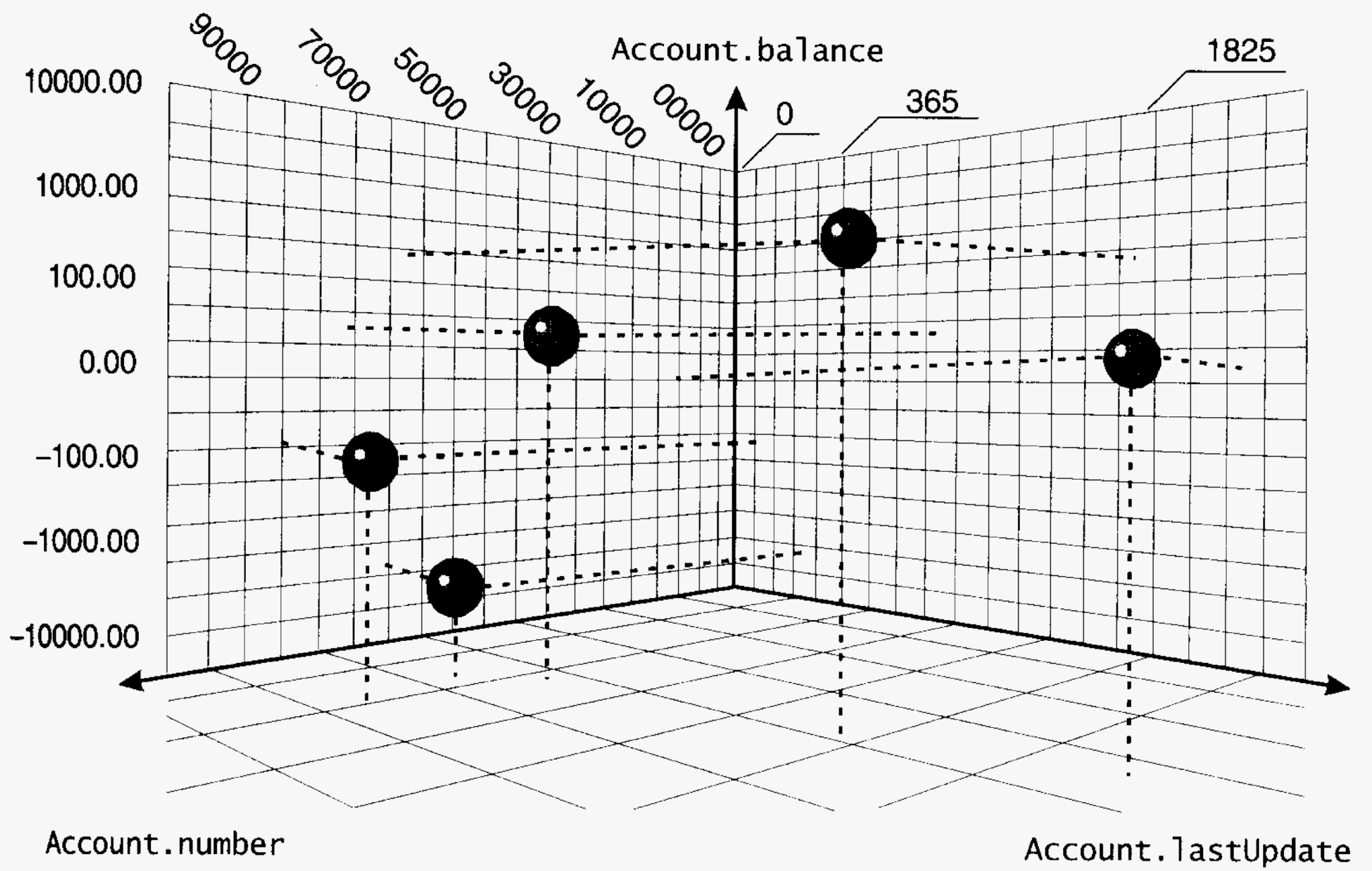


FIGURE 7.16 Primitive view of a state space.

Three abstract states

Shaded volumes

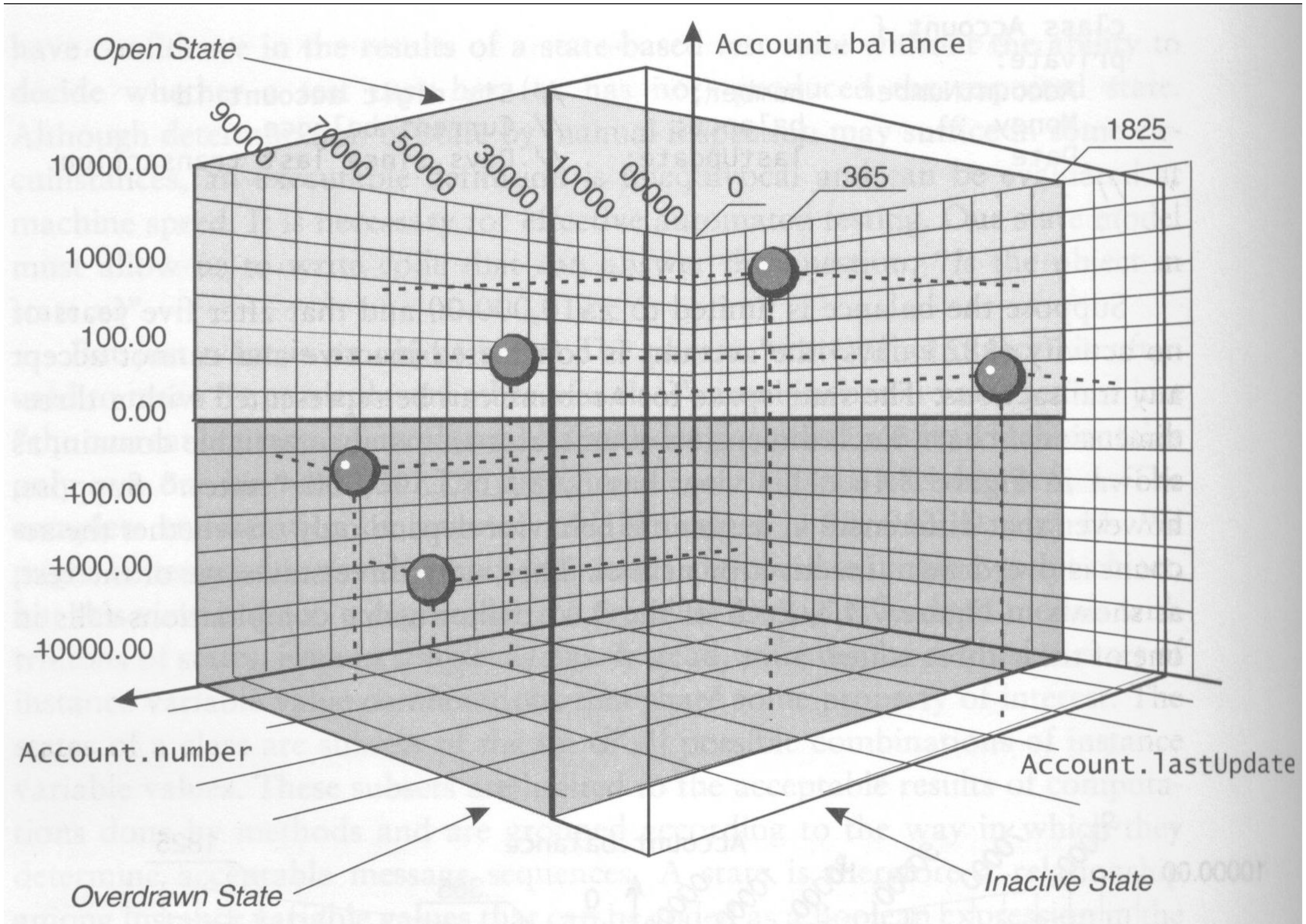


FIGURE 7.17 Abstract view of a state space.

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
  - `ensure a or b`
    in Eiffel this defines two states are possible

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**
  - **An optional action or actions**

Transition components

- An Event
  - 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
- A guard
  - Predicate associated with an event
  - No side effects
- An action
  - The side effects that occur

Alpha and Omega states

- The initial stage 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 **α state** just before construction
  - **α transitions go from α state to a constructor state**
- Similarly with **ω** and destruction (not necessary to model **ω** for languages that have garbage collection)
  - **ω transitions go from a destructor state to the ω state**