Introduction to Software Engineering

ECSE-321

Unit 11 – Mid-level State-Based Design

Mid-Level Design: Where Are We?

- Static class-based design
  - Appropriate for designing monolithic systems
- Interaction design
  - Appropriate for designing large distributed systems.
  - External interactions handled through interfaces.
  - Not sufficient for fine grain interactions (e.g., user interfaces)
- State-based design

Review of State Diagrams

- UML state diagram notation
- Illustrate uses of state diagrams
- Present heuristics for making good state diagrams

States, Transitions, Events

A **state** is a mode or condition of being.

A **transition** is a change from one state to another.

An **event** is a noteworthy occurrence at a particular time.

Finite Automata

- A formal model that abstracts everything about an entity except its states and state transitions
- A finite state machine or finite automaton specification must
  - Describe all states unambiguously (names)
  - Describe all transitions by stating the source and destination states and the triggering event
  - Designate an initial state

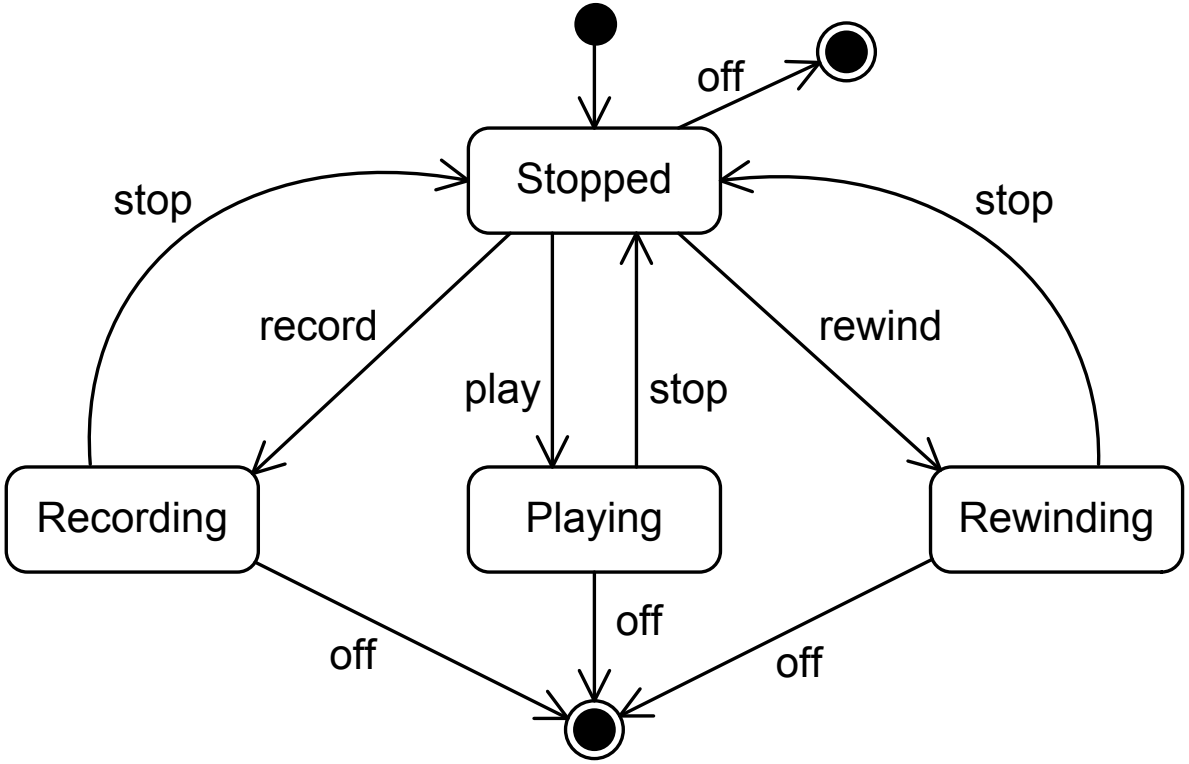
Determinism

- Finite automata may be ***deterministic***
  - Any event in any state triggers a transition to exactly one state
- Non-deterministic finite automata—any machine that is not deterministic
- We consider only deterministic machines

UML State Diagrams

- Represent states by rounded rectangles containing the state name
- Represent transitions by solid arrows labeled with one or more transition strings
- Transition strings
  - Describe triggering circumstances
  - Actions that result
- Initial pseudo-state designates the initial state
- Optional final state represents halting

State Diagram Example



State Transition String Format

event-signature guard / action-expression

- ***event-signature***—The empty string or an event name followed by a list of *event-parameters* enclosed in parentheses
  - parentheses may be omitted if there are no parameters
  - *parameter-name : type*
    - *parameter-name* is a simple name
    - *type* is a type description in an arbitrary format
    - *type* may be omitted along with the colon
- ***guard*** —a Boolean expression in square brackets
  - Format not specified in UML
- ***action-expression***—description of a computation done when the transition occurs
  - Format not specified in UML
  - Optional; if omitted, so is the slash

Transition String Examples

- buttonPress
- buttonPress(modifiers : Modifer[*])
- buttonPress / closeWindow
- buttonPress [enabled]
- buttonPress [enabled] / closeWindow
- [mode = active]
- / closeWindow
- buttonPress[enabled], mouseClick

State Diagram Execution Model

- The machine always has a current state.
- The machine begins execution in its unique initial state.
- When an event occurs:
  - If it matches an event-signature on a transition from the current state and the guard is absent or true, the transition occurs and the current state becomes the target state.
  - If it does not match an event-signature or the guard is false, no change of state occurs.

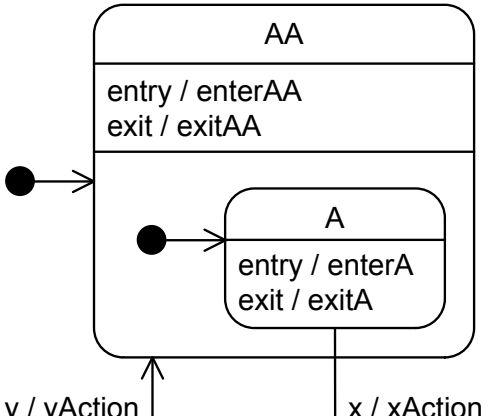
State Diagram Execution Model…

- If the computation occurring in the current state completes, and there is a transition from the current state with no event-signature:
  - If the guard is absent or true then the target state becomes the current state.
  - If the guard is false, no change of state occurs.
- If a transition has an action-expression, the action occurs when the transition takes place.
- If a final state becomes the current state, the machine halts.

State Symbol Compartments

As many as three compartments

- Name compartment
  - Optional; may be attached to a tab
  - Pathname
- Internal transitions
  - Internal transition specifications, one per line
  - Transition processed without causing a state change
  - Optional
- Nested diagram
  - Optional



Internal Transition Specifications

- May be a transition string or a string of the following form:

```
action-label / action-expression
```

  - action-label—one of the items from the table on the next slide
  - action-expression—description of a computation in a format not specified by UML

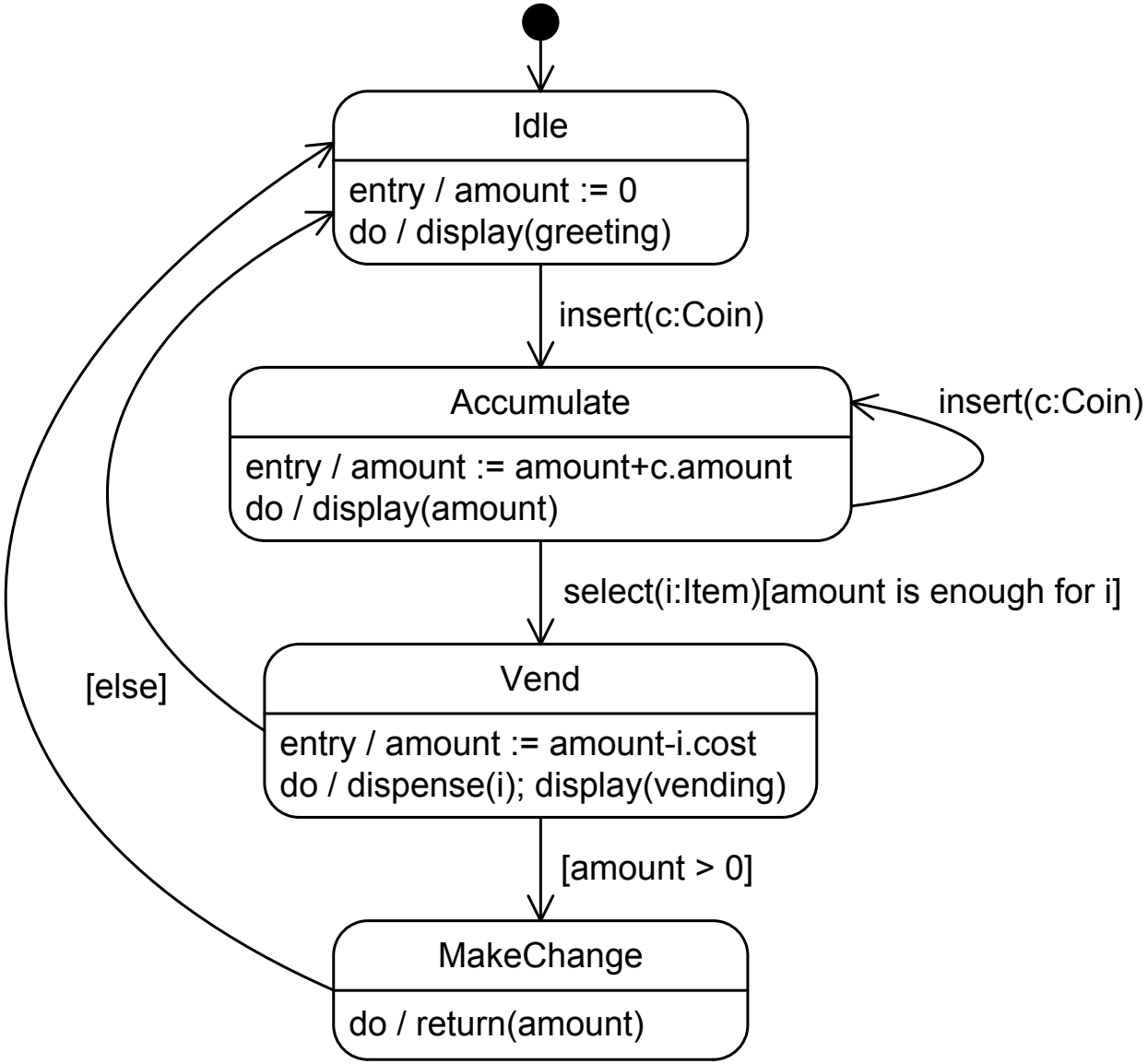
Internal Transition Action Labels

Action Label	Meaning
entry	Execute the associated *action-expression* (an *entry action*) upon state entry.
exit	Execute the associated *action-expression* (an *exit action*) upon state exit.
do	Execute the associated *action-expression* (a *do activity*) upon state entry and continue until state exit or action completion.
include	The *action-expression* must name a finite automaton. The named automaton is a placeholder for a nested state diagram (discussed below).

Internal Transition Examples

- buttonPress / beep
- keyPress(SPACE) / count++
- timeout [mode = alert] / displayAlertMsg
- entry / count := 0; sum := 0
- exit / ring bell
- do / display flashing light
- include / OrderProcessing

State Diagram with Internal Transitions Example



Nested State Diagrams

- A state without a nested state compartment is a **simple state**; one with a nested state compartment is a **composite state**

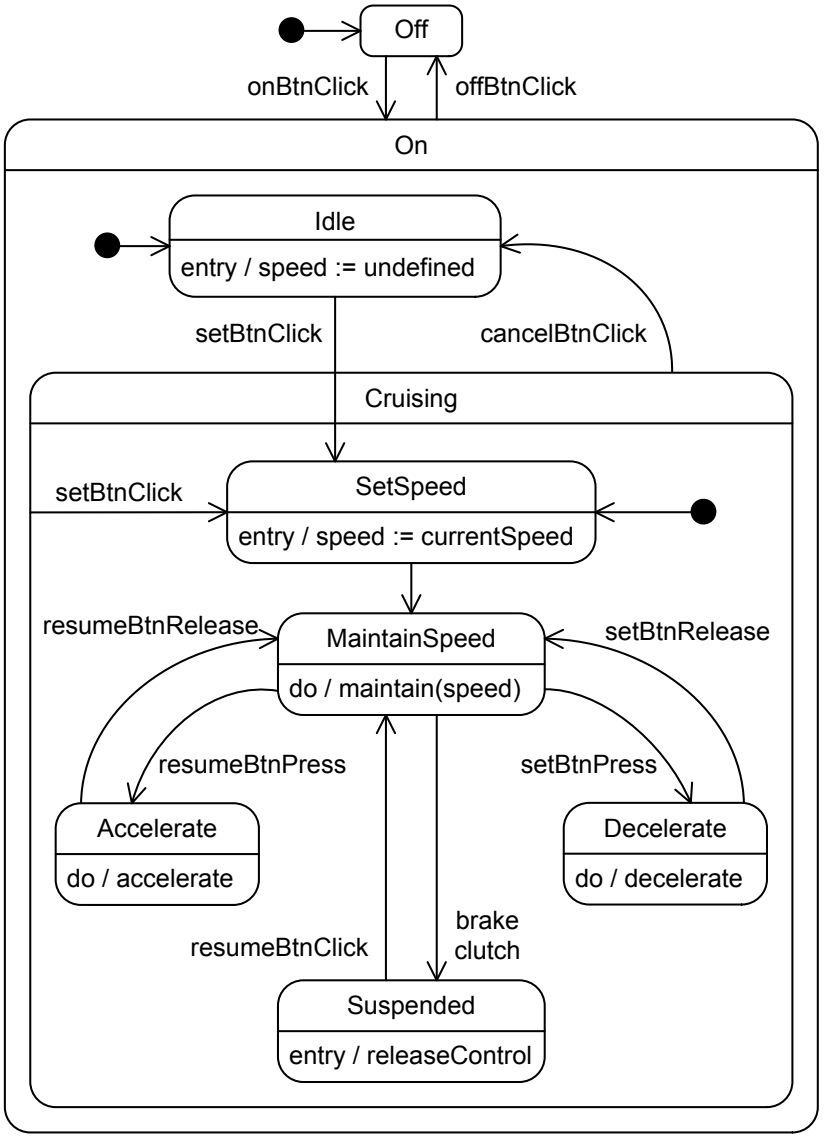
Types of Composite States

- *Sequential composite state*—The nested state compartment is a single region with *sub-states* or *inner states* and transitions.
- *Concurrent composite state*—The nested state compartment is comprised of two or more regions separated by *concurrent region boundary lines* (dashed lines) containing inner states and transitions.

Sequential Composite States

- When a sequential composite state is entered, so is one of its inner states (and likewise for further nested states), and they jointly become the current state.
  - Nested diagrams must have initial states or transitions must go directly to inner states
- When a sequential composite state is exited, so is the current inner state (and likewise for further nested states).

Sequential Composite States Example



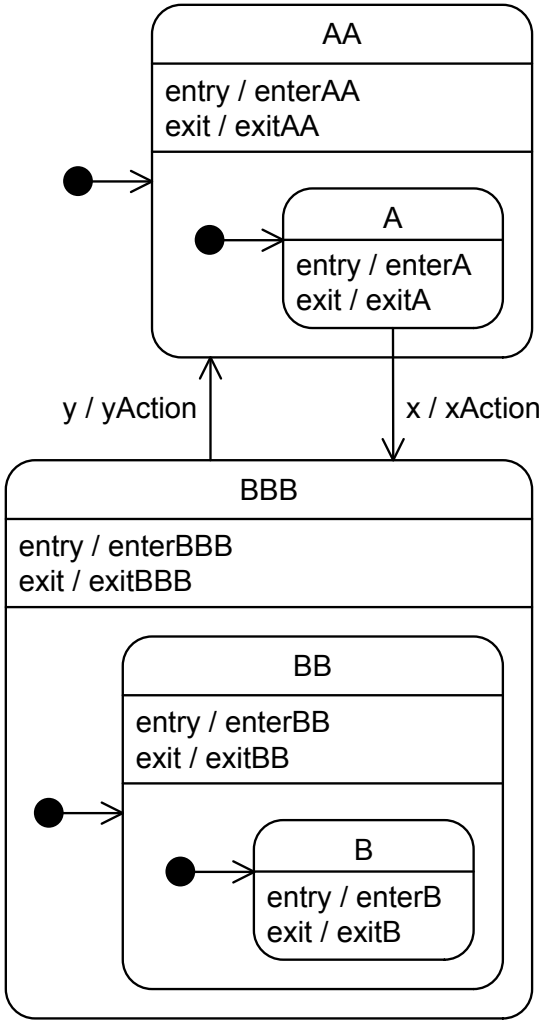
Action Execution Order

When an initial state is entered, the state's entry actions are executed, followed by the entry actions of any initial sub-states.

When a transition causes exit from simple state A and entry to simple state B

1. Simple state A's exit actions are executed;
2. The exit actions of any exited composite states enclosing A are executed, in order from innermost to outermost exited states;
3. The transition action is executed;
4. The entry actions of any entered composite states enclosing B are executed, in order from outermost to innermost entered states; and
5. Simple state B's entry actions are executed.

Action Execution Order Exercise

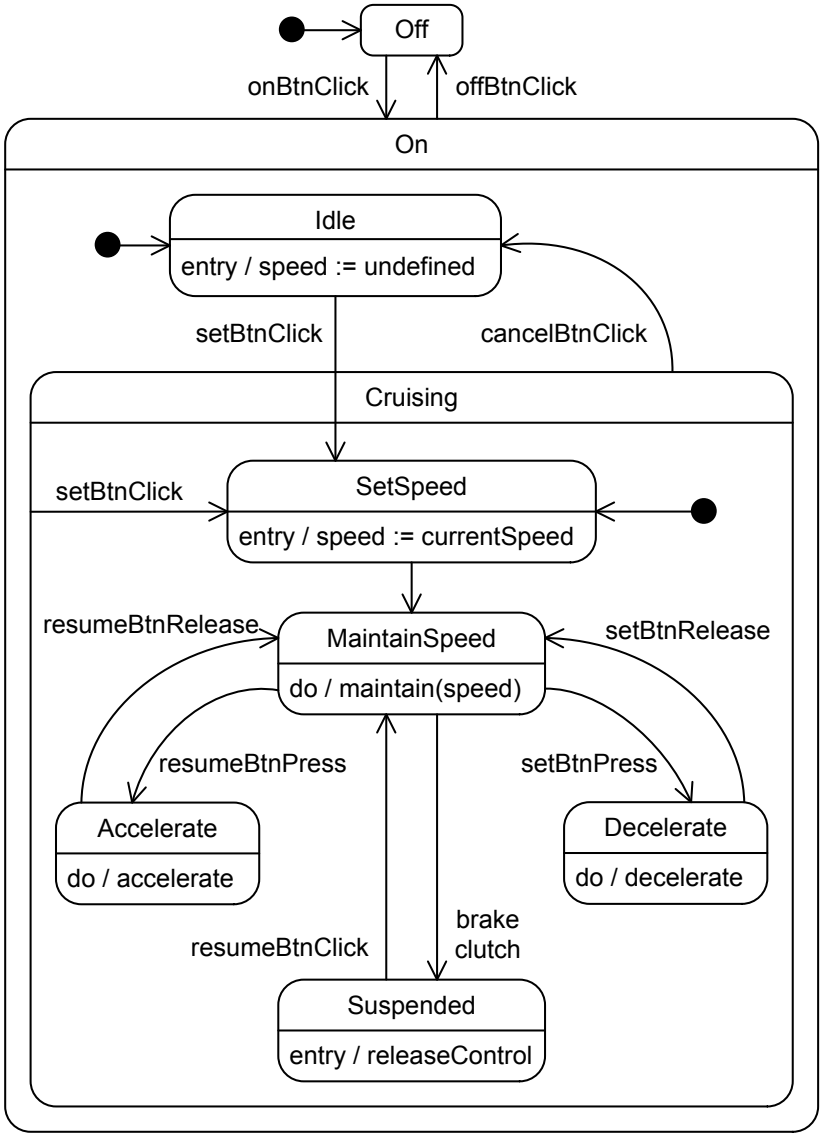


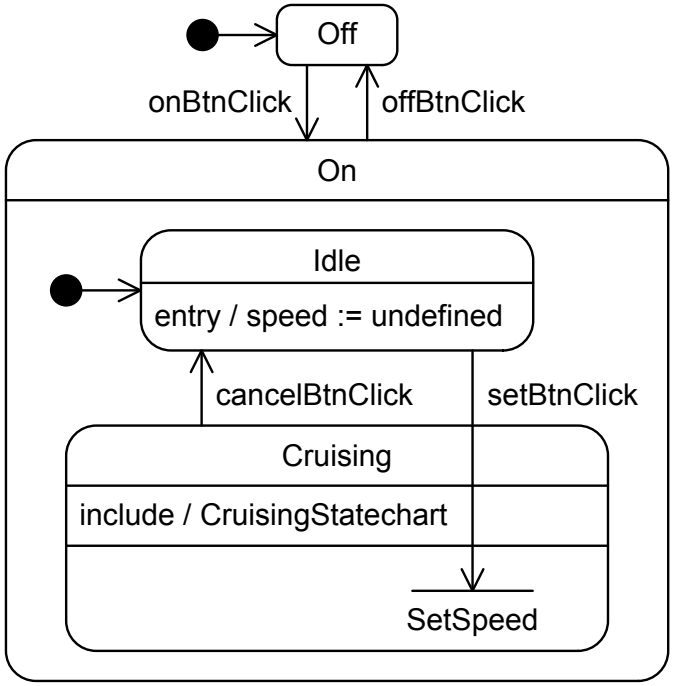
When event x occurs, what are the action execution order?

Stubbed States

- Stub symbol used as a termination and origin point for transitions to and from states in a suppressed nested state diagram
- A *stub* or *stub symbol* is a short line labeled with the suppressed state name

Stubbed State Example





Stubbed State Diagram

Using Sequential Composite State

- Any finite automaton can be described by a state diagram with only simple states.
- Sequential ***composite states*** simplify state models in two ways:
  - Organize states into hierarchies
  - Consolidate many transitions

Sequential State Diagram Heuristics

- Check that no arrow leaves a final state.
- Check for ***black holes*** (dead states) and ***white holes*** (unreachable states).
- Label states with adjectives, gerund phrases, or verb phrases.
- Name events with verb phrases or with noun phrases describing actions.
- Name actions with verb phrases.

Sequential State Diagram Heuristics…

- Combine arrows with the same source and target states.
- Use stubs and the include internal transition to decompose large and complicated state diagrams.

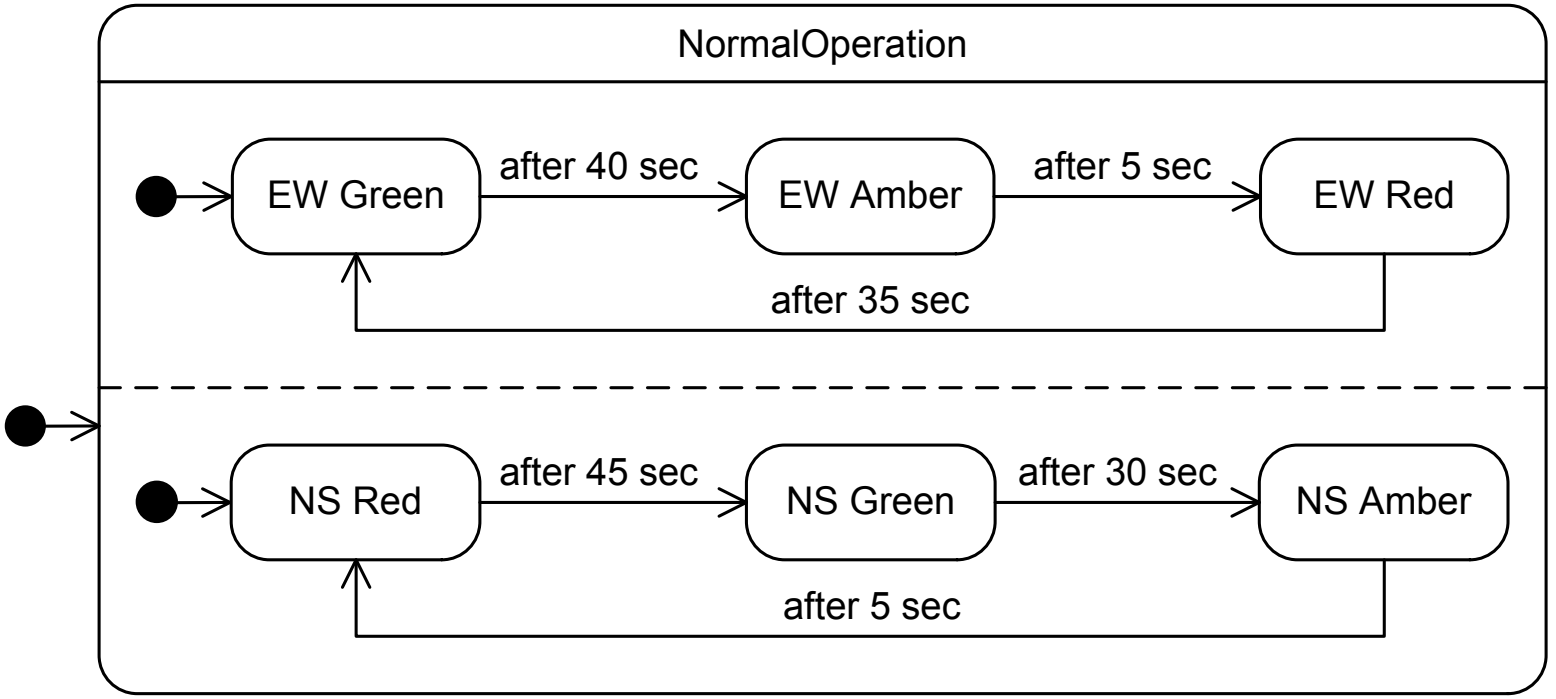
Sequential State Diagram Heuristics…

- Make one initial state in every state diagram (including nested state diagrams).
- Check that no event labels two or more transitions from a state.
- Check that all guards on the same event are exclusive.
- Use [else] guards to help ensure that guards are exclusive and exhaustive.

Concurrent Composite States

- The regions in a concurrent composite state nested state compartment contain state diagrams that execute in parallel.
- One state in each region is entered when the concurrent composite state is entered.
- One state from each region is always among the joint concurrent states until the concurrent composite state is exited.
- Events cause transitions in each concurrent region to occur simultaneously.

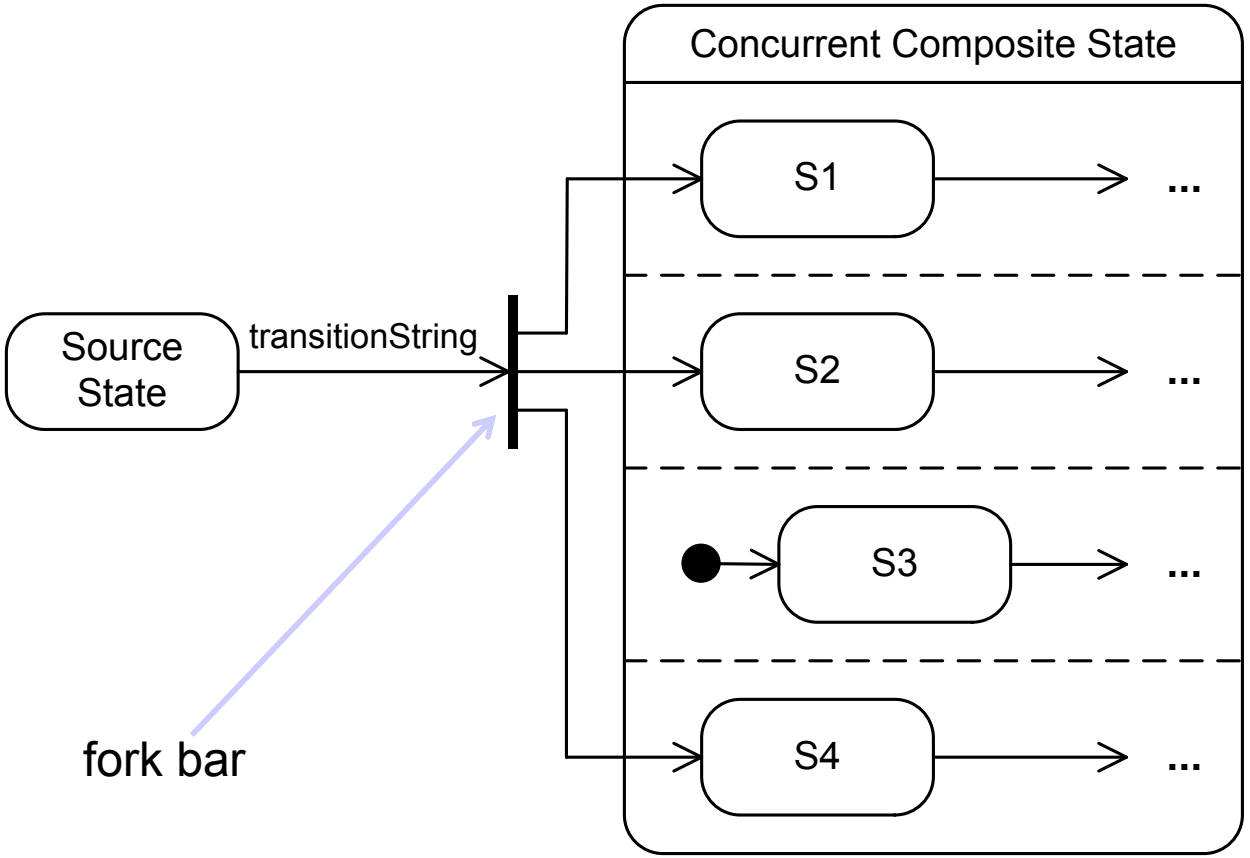
Concurrent Composite State Example



Entering Concurrent Composite States

- Make a transition to the concurrent composite state boundary
  - The initial state in each region becomes the current state
- Make a transition to individual states in different regions
  - Main transition goes to a fork bar
  - Transitions to individual state come from the fork bar
  - A region without a state targeted by a transition begins in its initial state

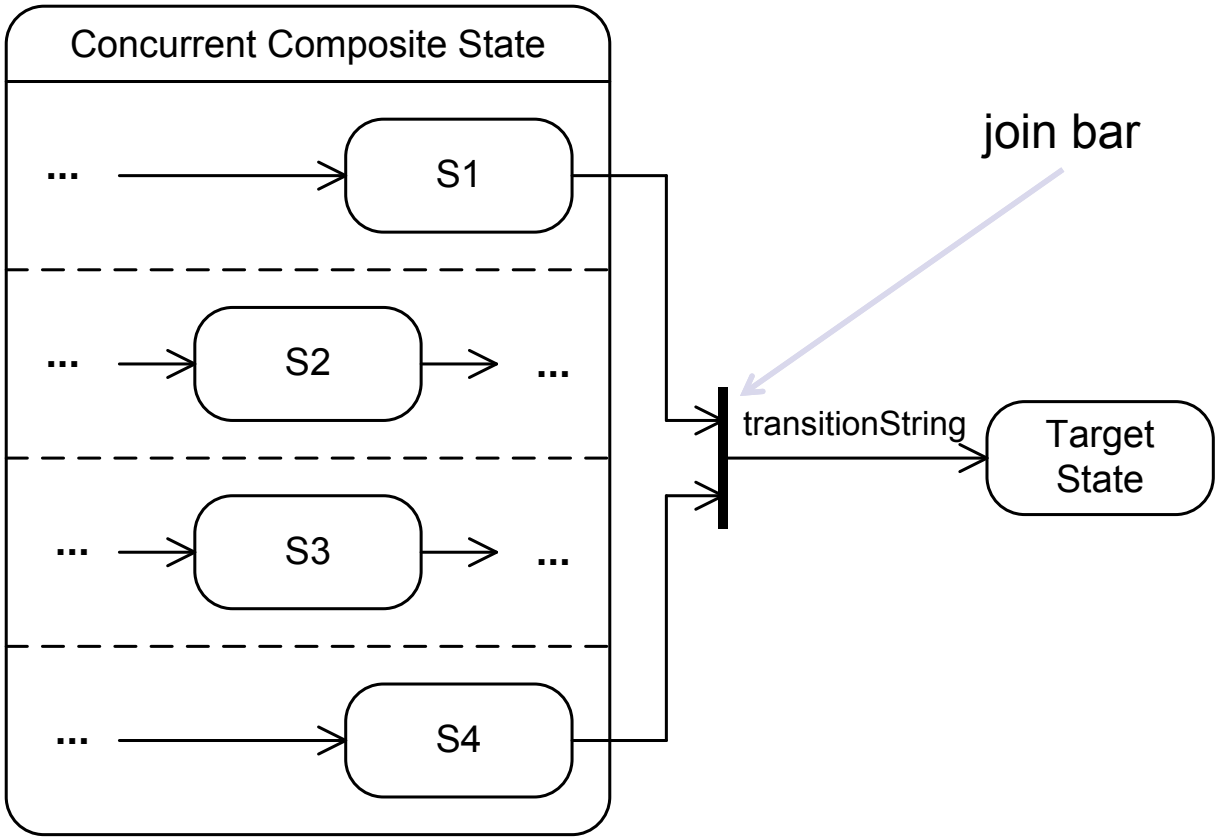
Entering Selected Concurrent States: Illustration



Leaving Concurrent Composite States

- Make a transition from the concurrent composite state boundary
  - For a non-completion transition, all concurrent sub-states are exited immediately
  - For a completion transition, the current state must be a final state in every concurrent region
- Make a transition from one or more concurrent sub-states
  - Coordinated transitions can go to a join bar
  - All other sub-states are exited immediately

Leaving Selected Concurrent States: Illustration



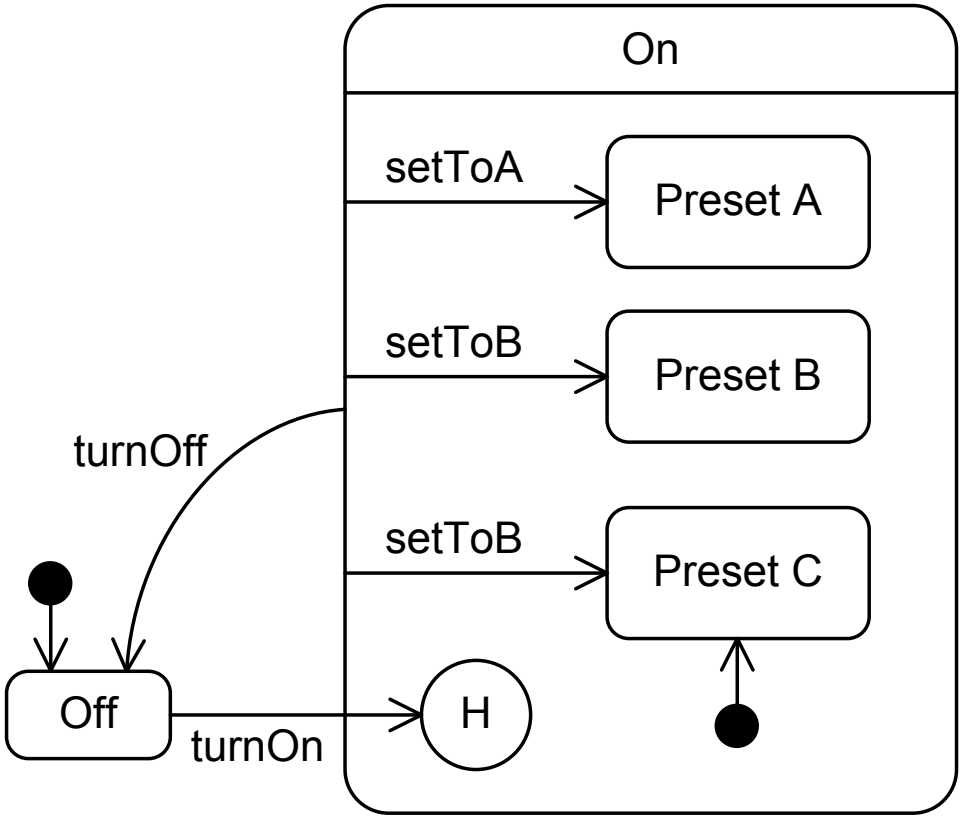
Using Concurrent Composite States

- Any concurrent composite state can be represented by a diagram with only simple states, but it will have ***many more states than the concurrent composite*** state.
- Concurrent composite states thus simplify diagrams.
- On the other hand, diagrams with ***concurrent composite states are often hard to understand***.

History States

- A *history state* is a pseudo-state indicating that the sub-state last active when a composite state was exited should be reentered.
  - Symbol is a circled H
- Many common devices have persistent state, so this is a useful modeling feature.

History State Example



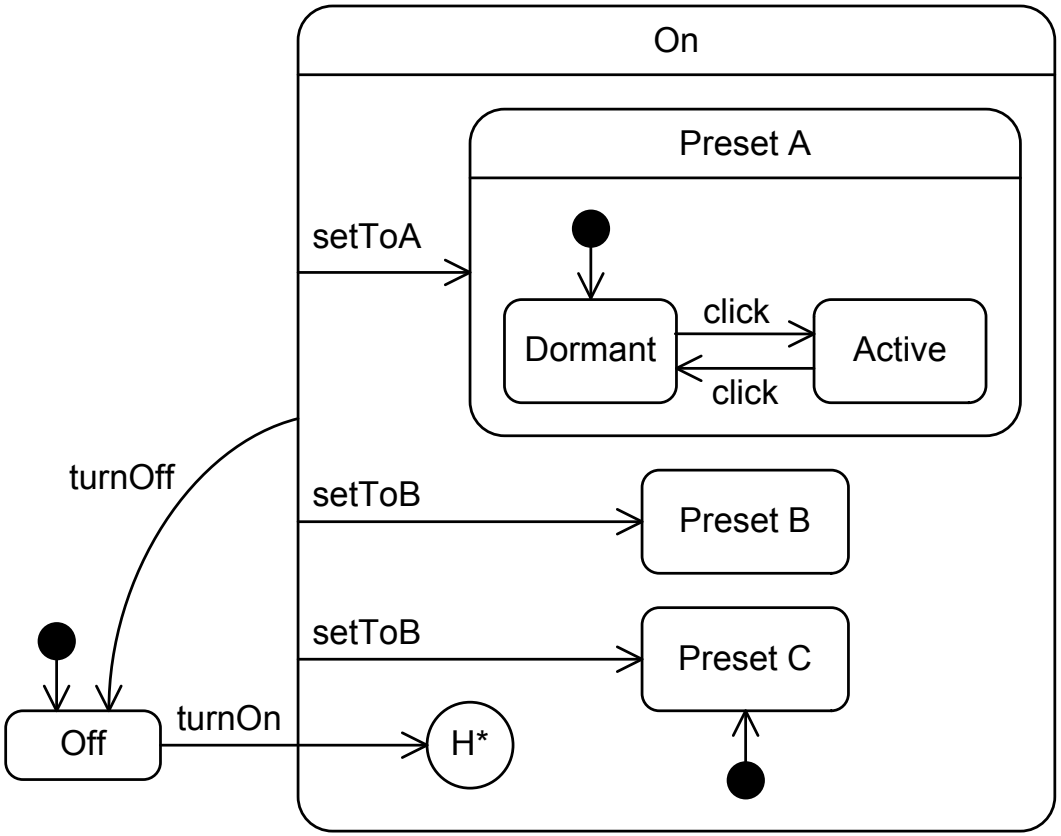
History State Restrictions

- May only appear in a region of a composite state
- Transitions may only enter a history state from outside the composite state
- May have at most one unlabeled outgoing transition to a peer state
  - Indicates the default reentered state if the composite state has not yet been entered
- History states may not have internal transitions, nested compartments, etc.
- History state is forgotten if the current inner state becomes a final state.

Deep History States

- A history state indicates reentry to a state at the same nesting level.
  - States at lower nesting levels are entered as usual (initial states).
- A *deep history state* is a pseudo-state indicating that the states last active at every nesting level when a composite state was exited should be reentered.
  - Symbol is a circled H*

Deep History State Example



More State Diagram Heuristics

- Designate an initial state in every concurrent region of a concurrent composite state.
- Check that transitions to several concurrent sub-states go through a fork.
- Check that arrows connected to transition junction points are properly labeled.
- Check that at most one unlabeled arrow emanates from each history state.

Summary

- State diagrams are a powerful UML notation for showing how entities change over time.
- Entity states are represented by rounded rectangles, and state changes by labeled transition arrows.
- Transition strings allow specification of transition in terms of events and environmental conditions, and allow specification of transition actions

Summary…

- Sequential composite states provide a means to show state hierarchies.
- They also allow (sometimes radical) reductions in the number of transitions, simplifying models.

Summary…

- State diagrams can show concurrency concurrent composite states, but these are governed by somewhat complex rules and hard sometimes hard to understand.
- Compound transitions allow combination of several transitions with common transition strings into one.
- History and deep history states allow state diagrams to model persistent states.

Designing with State Diagrams

- Recognizers (acceptors) and transducers
- Uses of recognizers and transducers
  - Devices
  - Lexical analyzers
- Dialog maps and user interface diagrams

Kinds of Finite Automata

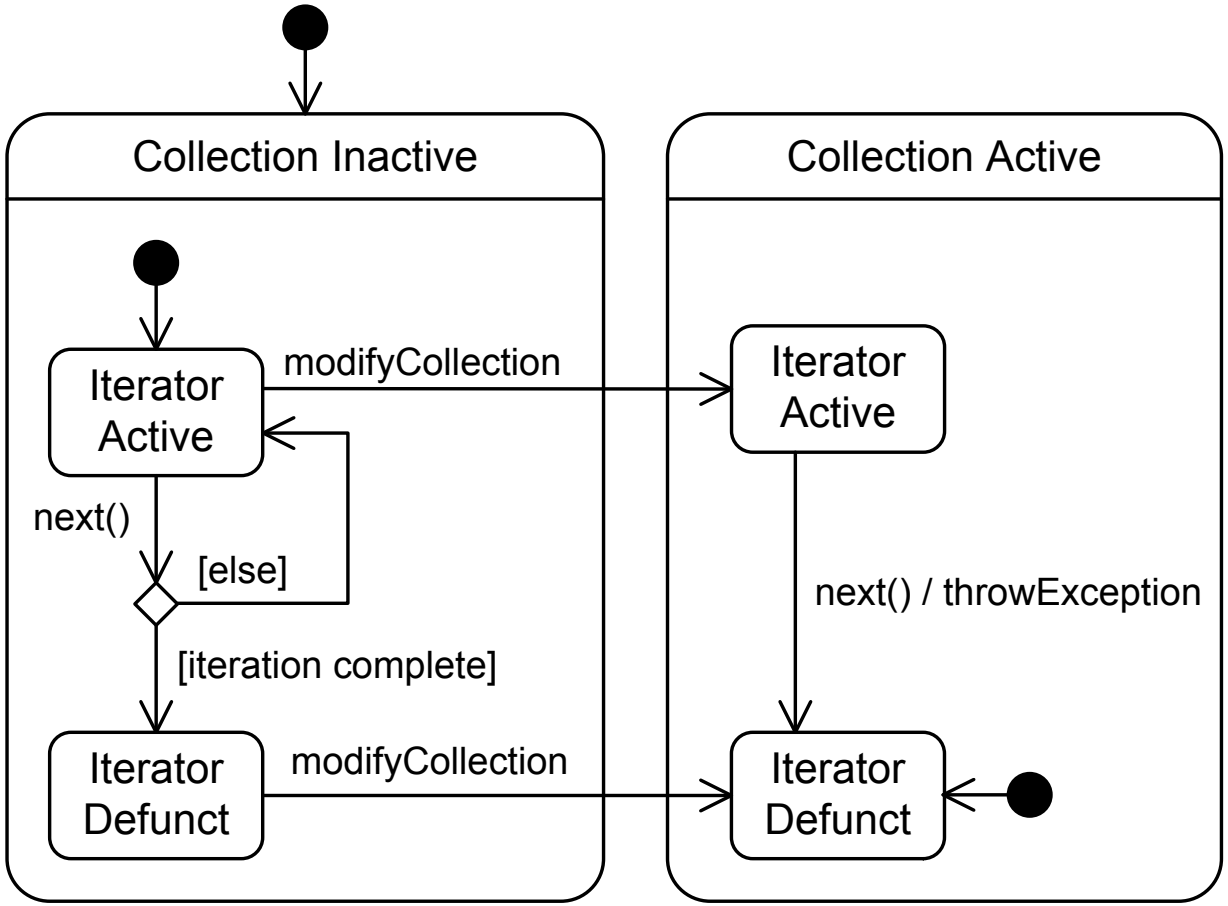
An **acceptor** or **recognizer** is a finite automaton that responds to events but generates no actions.

A **transducer** is a finite automaton that both responds to events and generates actions.

Automaton Uses

- Acceptors or recognizers are used to determine whether input is valid (accepted or recognized).
  - Accepted if and only if the machine is in an accepting state when the input is consumed
  - Examples: translators, interpreters
- Transducers are used to model things that transform inputs to outputs.
  - Examples: devices, programs with complex state-based behavior

Transducer Example



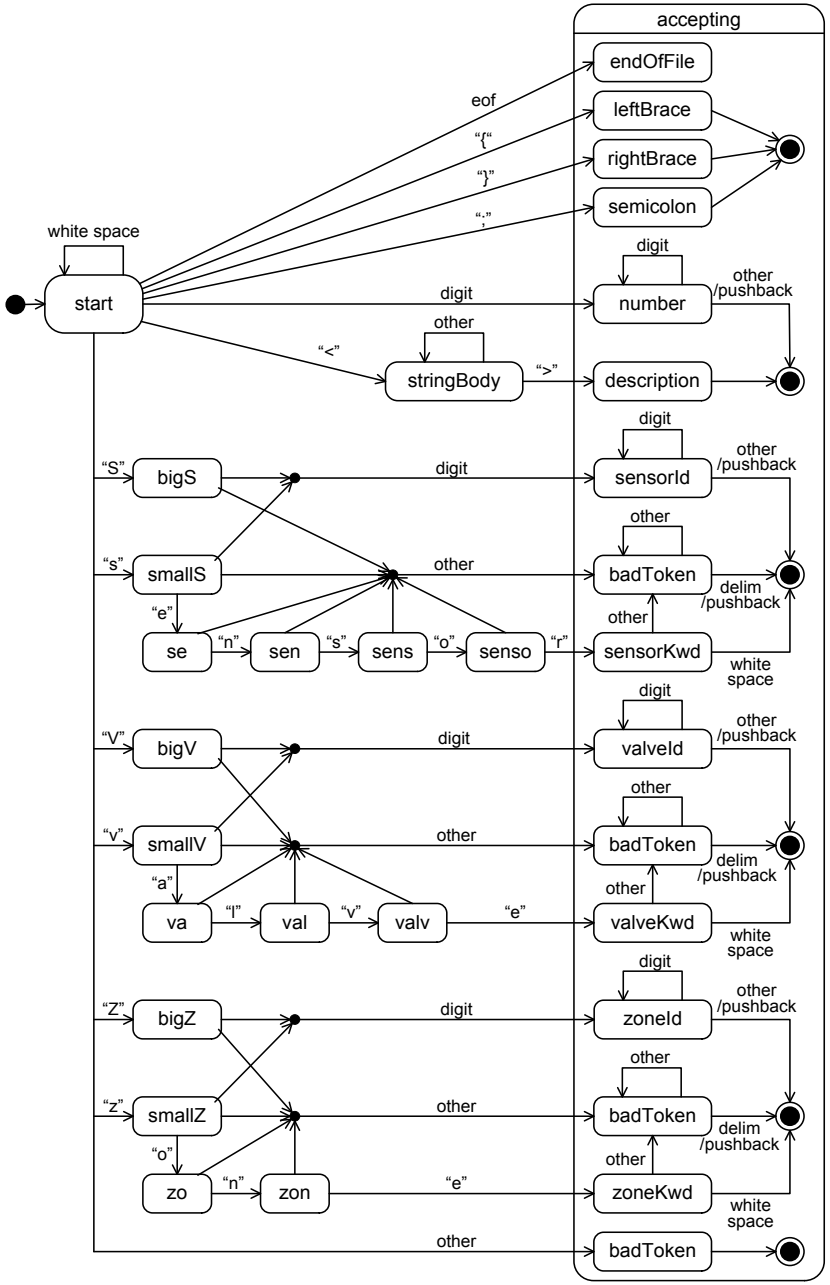
Acceptor Example

- A *lexical analyzer* is a program unit that transforms a stream of characters into a stream of tokens.
  - *Token*: a symbol recognized by a program
- Example: The Irrigator configuration file requires certain tokens.

Irrigator Configuration File Tokens

Token Name	Token Description
endOfFile	End of the input file marker
leftBrace	{
rightBrace	}
zoneId	“Z” or “z” followed by one or more digits
sensorId	“S” or “s” followed by one or more digits
valveId	“V” or “v” followed by one or more digits
Semicolon	“;”
zoneKwd	The keyword “zone”
sensorKwd	The keyword “sensor”
valveKwd	The keyword “valve”
number	One or more digits
description	Characters between “<” and “>”

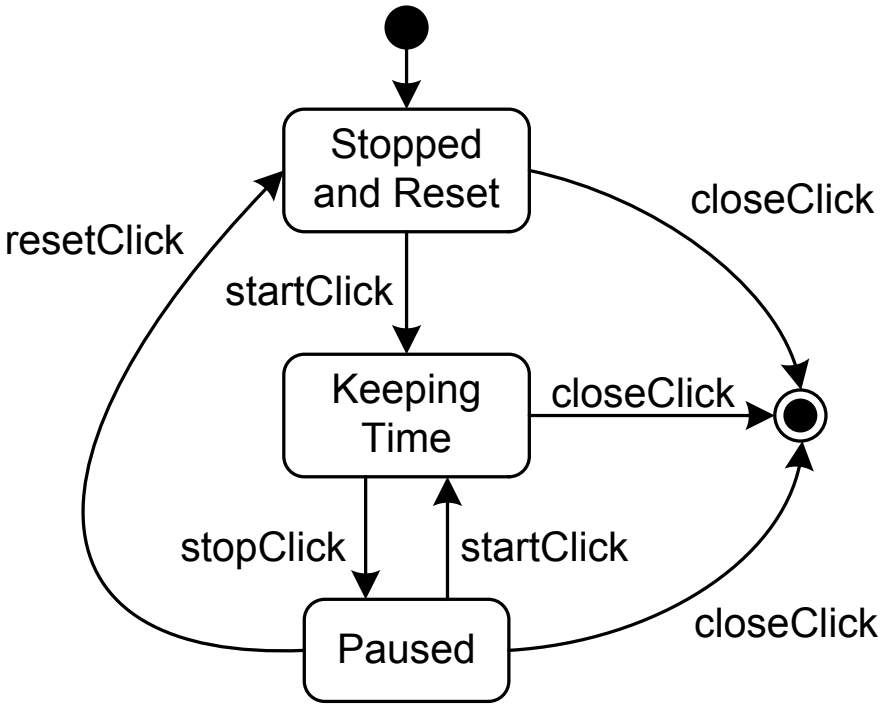
Lexical Analyzer for the Irrigator front end



Dialog Maps

- Acceptors are also used to model user interfaces.
- A *dialog map* is a state diagram whose nodes represent user interface states.
- Events are occurrences (usually user input actions) that drive the program between user interface states.

Dialog Map Example

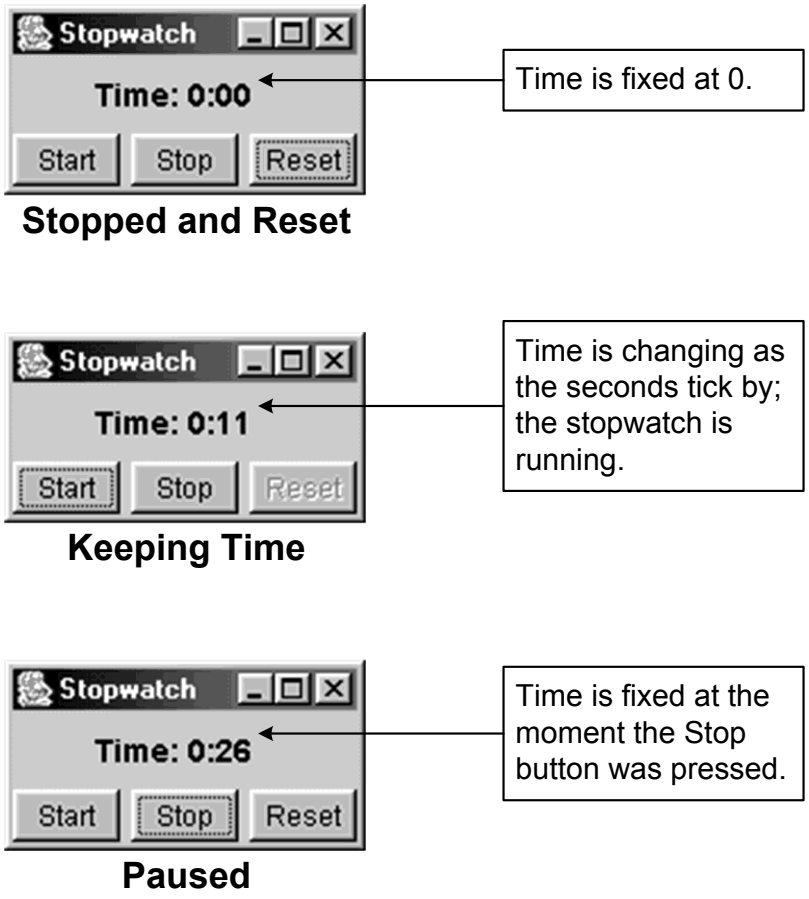


User Interface Diagrams

- A *user interface diagram* is a drawing of (part of) a product's visual display when it is a particular state.
- Dialog maps and user interface diagrams can be used together:

  Every user interface diagram should specify the visual form of a state in a dialog map, and every state in a dialog map should have its visual form specified by a user interface diagram.

User Interface Diagram Example



Summary

- Acceptors or recognizers are finite automata used to test input validity; transducers are finite automata that transform inputs to outputs.
- State diagrams model both sorts of automata.
- Dialog maps are acceptors whose states represent user interface states; user interface diagrams are drawings of visual displays.
- User interface diagrams and dialog maps are used in conjunction to model user interfaces.