State Machines

Dr. Constantinos Constantinides, P.Eng.
Computer Science and Software Engineering
Concordia University

What is a state machine?

Overloaded terminology. An attempt to find order

- A state machine (also: finite-state machine, finite (state) automaton), is a mathematical model of computation, and it has two representations:
  - Mathematical, and
  - Visual.
- The visual representation is captured by a state transition diagram (also: State [machine] diagram, statechart [diagram]).

Event-driven (reactive) systems

- We are interested in systems whose behavior is determined by events, e.g. user actions, signals received by sensors, time signals, or messages from other systems, etc.

- We refer to these systems as event-driven (or reactive).

Event-driven (reactive) systems /cont.

- An event-driven system can be at one of several conditions under which it satisfies some invariant property.

- Such conditions are called states.

- Upon receiving an event, the system reacts.

- The response to an event generally depends on both the type of the event and on the state of the system.

- This response may include a state transition leading to a change of state, or it may include some action undertaken by the system.

Event-driven (reactive) systems /cont.

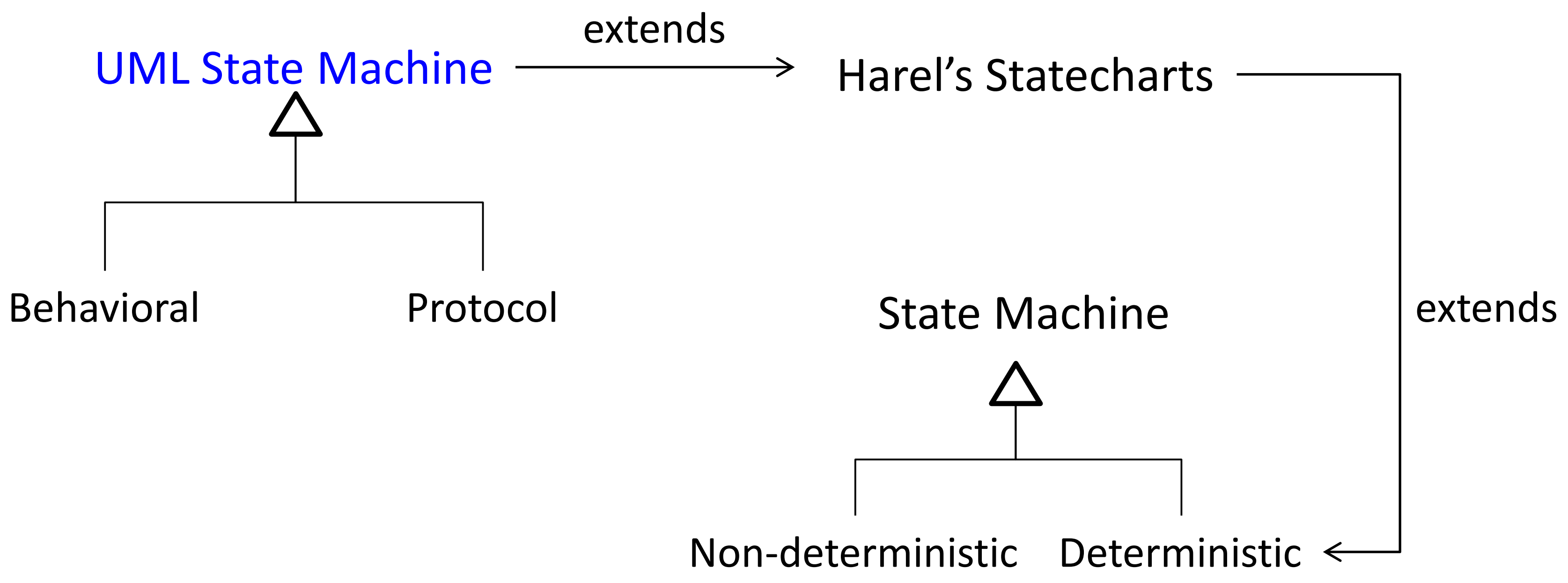
- The pattern of events, states, and state transitions among those states can be abstracted and represented as a finite-state machine (FSM).
- An FSM models the behavior of a system and describes its life-cycle.
- This discussion is on a variant of finite state machines adopted by the Unified Modeling Language (UML) – see next.

Some historical background

- Finite-state machines were originally introduced in 1962 by Gill.
- In 1984 Harel proposed statecharts as a significant extension over traditional machines.
- A statechart is a formalism to model the dynamic behavior of a component at any level of abstraction like e.g. an object, a system unit, a use case, or the entire system itself.
- The Unified Modeling Language adopted Harel's statecharts in its specification and extended them.
- In this course, we study state machines under the OMG UML specification(*), referred to in the literature as UML state machine (or UML statechart) and referred to throughout this presentation as state machine.

(*) www.omg.org/spec/UML/2.5.1/PDF

Some historical background /cont.



PART 1: Behavioral State Machines

Behavioral state machines

- A behavioral state machine M is formally defined as a tuple.

  $M = (Q, \Sigma_1, \Sigma_2, q_0, V, \Lambda)$,

  where
- Q: A finite, non-empty set of states.
- Σ_1: A finite, non-empty set of events.
- Σ_2: A finite set of actions.
- $q_0 \in Q$: The initial state (or start state).
- V: A set of state variables.
  - Every state variable $v \in V$ is a global variable and can be accessed at every state $q \in Q$.
- Λ is a finite set of transitions.

State machines /cont.

- A transition is a relationship between two states.
- A transition indicates that when an event occurs, perhaps under some condition called a (transition) guard, the entity changes from the source state to the target state.
- Additionally, upon a state transition an action may execute.
- All parts of a transition label are optional.

Transitions

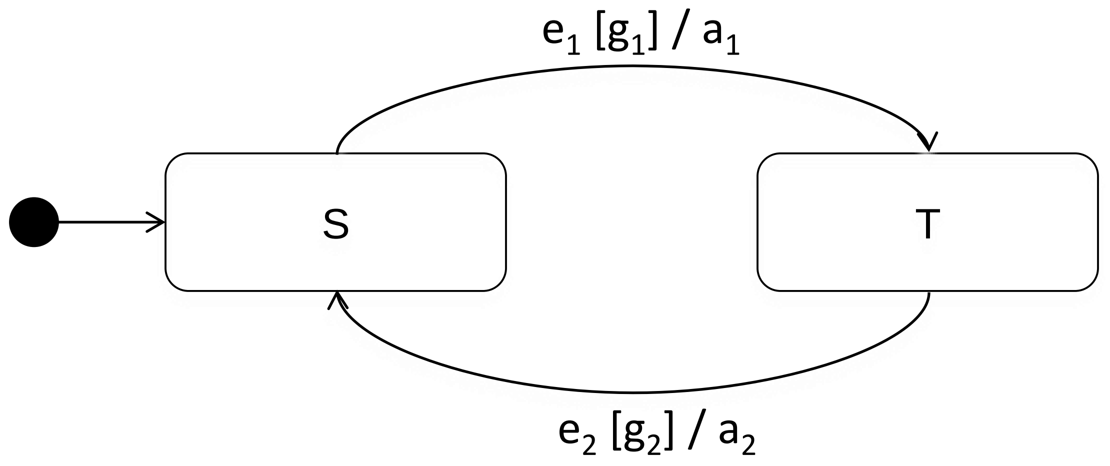
- Formally, a transition is described as

  $\lambda \in \Lambda$ is $q \xrightarrow{e\,[g]\,/\,a} q'$

  where
- $q, q' \in Q$ are the source and target states respectively,
- $e \in \Sigma_1$ is an event,
- g is a guard, and
- $a \in \Sigma_2$ is an action.
- A variable $v \in V$ affected in the transition is denoted as v' in state q'.

Visualization of a state machine

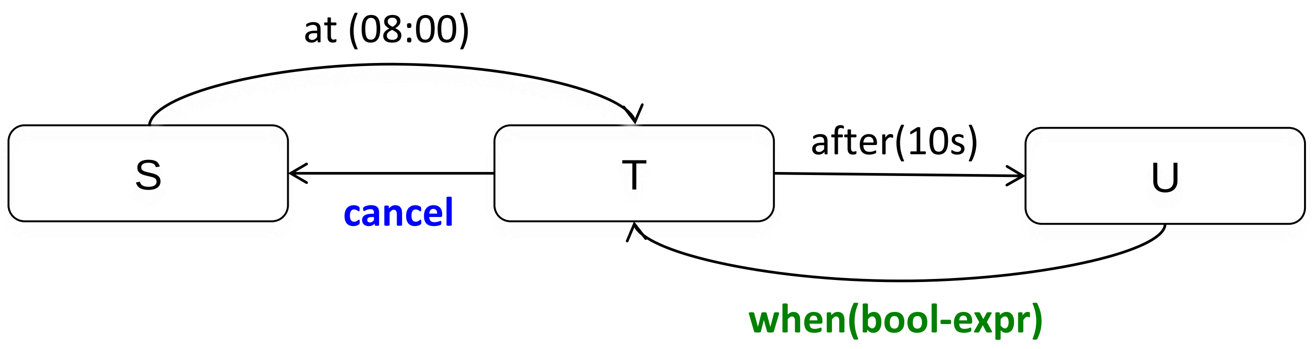
- A state machine can be illustrated by a directed graph, where the nodes represent the states and the substates, and where the edges represent the transitions.



- The machine's underlying behavior is modeled as a traversal of this graph.

Types of events

- There are four types of events that can trigger a transition:

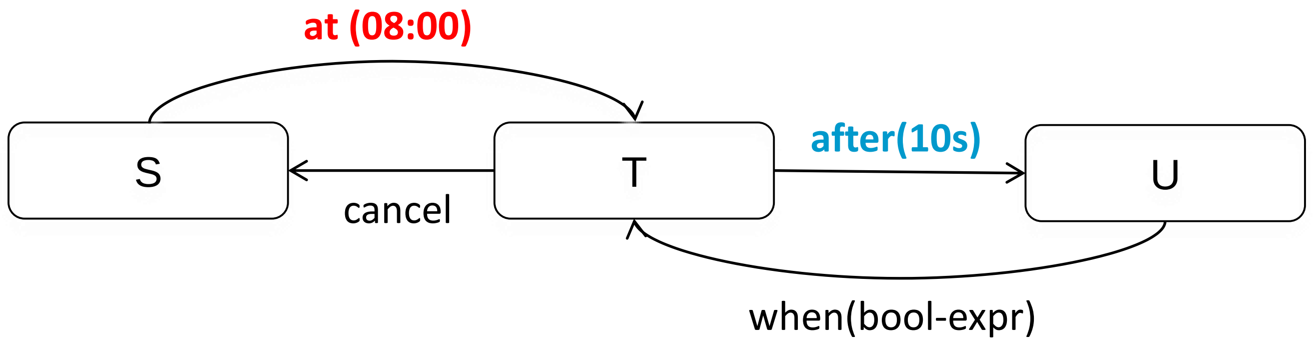


- **Call event**: An external request, e.g. cancel.

- **Change event** (Makes use of keyword **when**): A condition that will trigger a transition when true.

Types of events /cont.

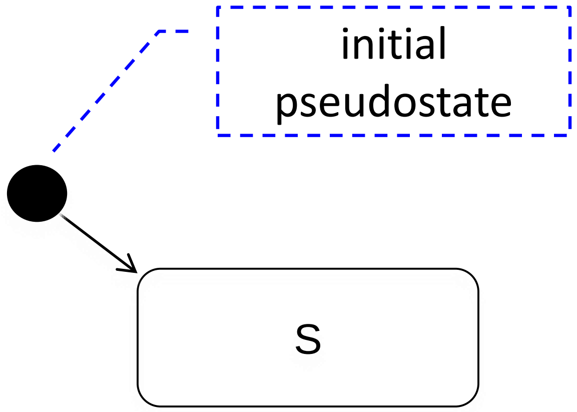
- There are four types of events that can trigger a transition:



- **Signal event** (Makes use of keyword `at`): Triggered by a (internal or external) clock, e.g. `at(08:00)`.

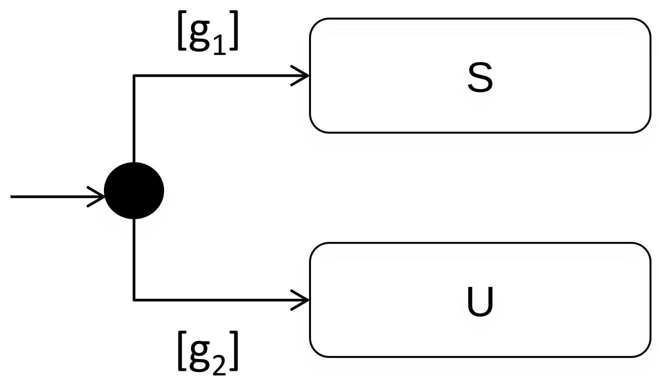
- **Time event** (Makes use of keyword `after`): When the source state has been active over the specified length of time, the guard (if present) is evaluated and a transition occurs if the guard is true. If no guard is present, then a transition occurs, e.g. `after(10s)`.

Pseudostates and the Initial pseudostate



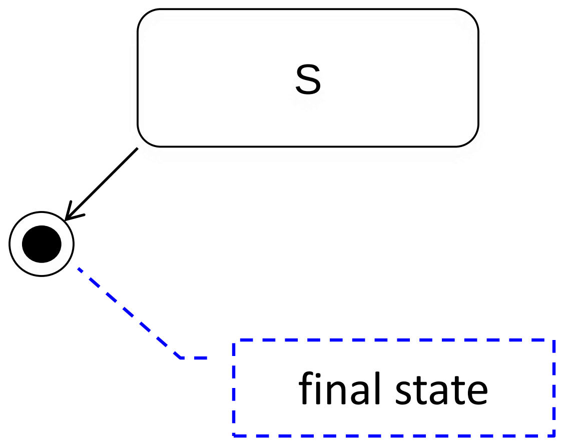
- A pseudostate is used to combine and direct transitions.
- The Initial pseudostate marks the starting point of a region that leads to the region's default state.
- The Initial pseudostate may optionally have a name.

The Initial pseudostate /cont.



- When more than one transition originates from the initial pseudostate, then the starting state is determined by the evaluation of the guard on each transition.
- In this example, either of S or U can be the system’s initial state, depending on the evaluation of $g_{1,2}$.

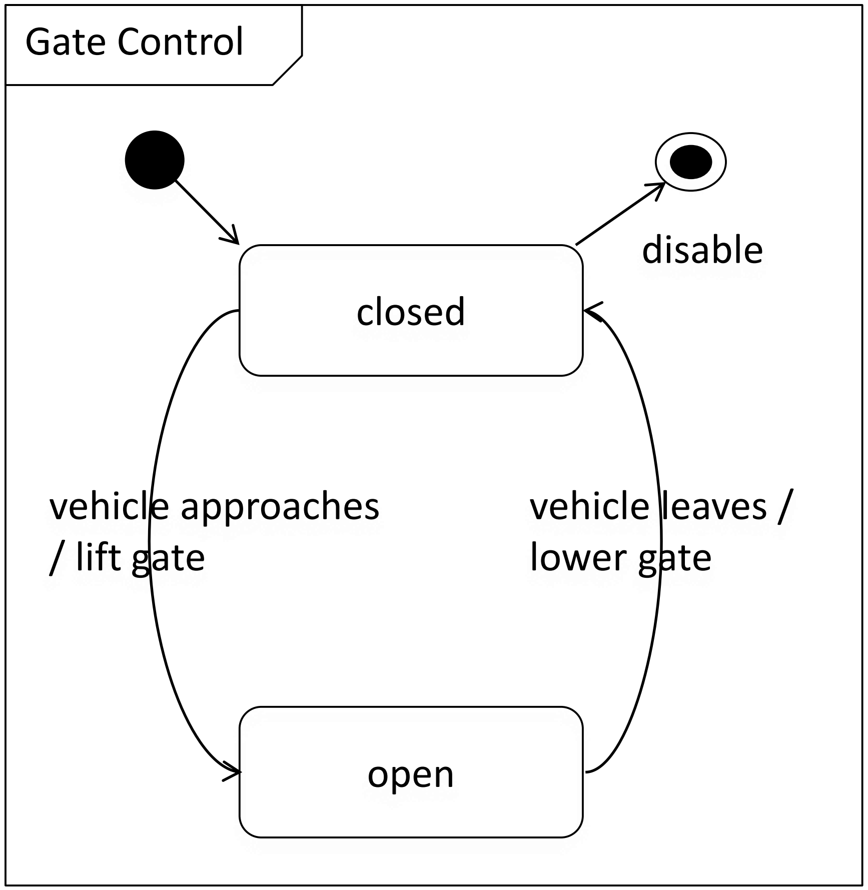
The Final state



- The final state (or exit state) is a special state that indicates that the enclosing region is completed.
- Note that the final state is not considered a pseudostate.
- The final state may optionally have a name.

Example: Gate Control

States, events, actions, and transitions



- Initially the system is at state closed.
- Once a vehicle approaches the gate, a sensor would produce an event.
- In response, the system lifts the gate while performing a transition to state open.
- While at state open, once the vehicle leaves from the gate, another sensor would produce an event.
- In response, the system lowers the gate while performing a transition to state closed.

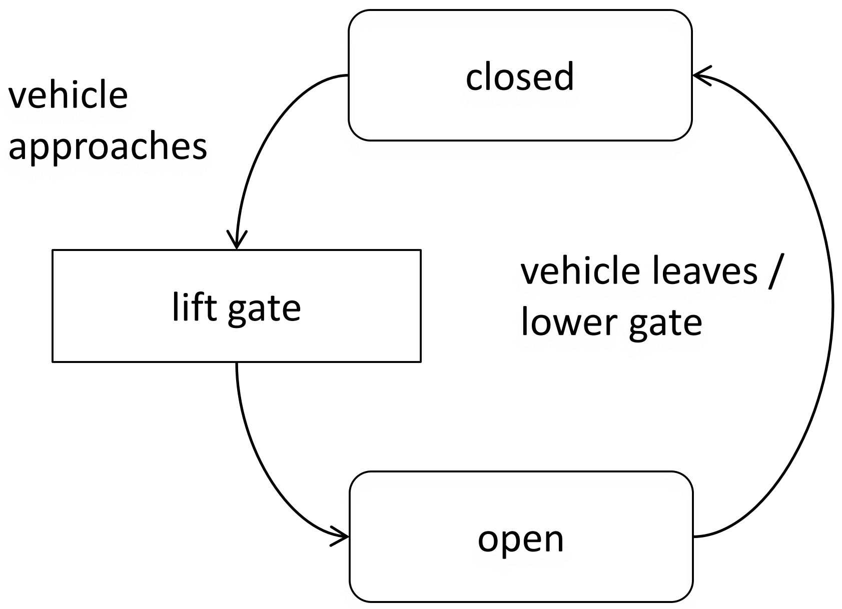
Event-driven (reactive) systems revisited

- When a state machine is <u>executed</u>, it enters the initial state, and adopts the behaviour associated with that state.
- When a machine is *in* some state, this implies that the machine behaves in the way that state describes.
- As the machine reacts to events, it performs transitions to other states, thus changing its behaviour over time.

Event-driven (reactive) systems revisited /cont.

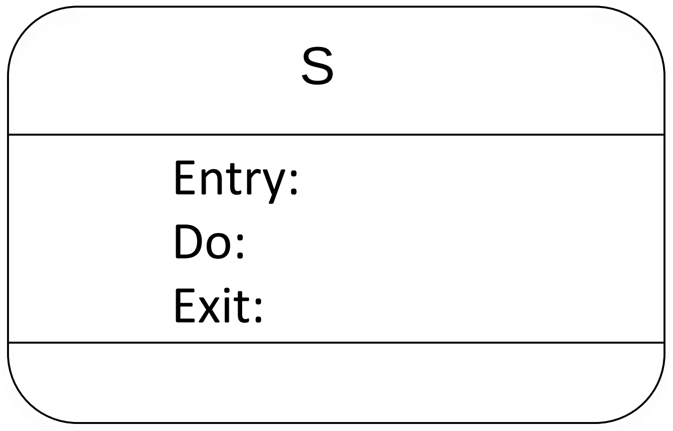
- A state machine need not illustrate every possible detail.
- For example, if an event arises that is not represented in the model, the event is ignored at that particular level of abstraction.
- We can create machines which describe the life-cycle of a system at any simple or complex level of detail, depending on our needs.
- During software development, a state machine may be deployed to model the dynamic behaviour of any real or conceptual element at any level of abstraction, e.g. object, use-case, the entire system, system sub-units etc.

Actions



- Actions can be denoted by a slash (/), or within a rectangle.

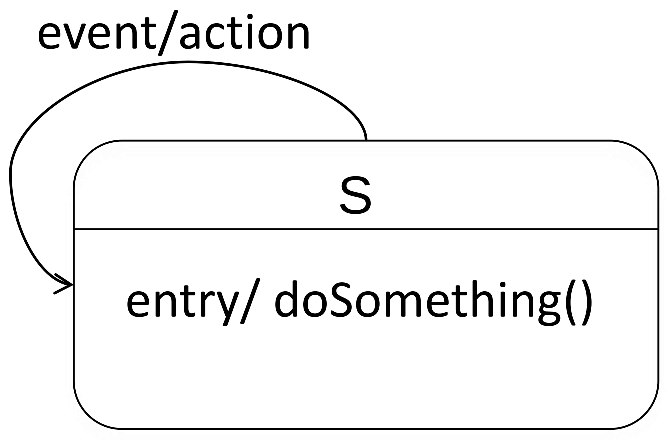
State behavior



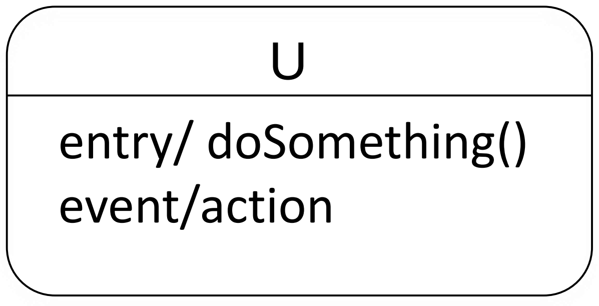
- A state may optionally contain its own behavior:
- Entry behavior: Executes immediately upon entering a state.
- Exit behavior: Executes immediately before exiting the state, and
- Do behavior: Executes while a state is active.

Recursive and internal transitions

- Internal transitions are denoted within the state, like internal activities.
- A recursive transition is one where the same state acts as the source and target, and both its entry and exit behavior are executed. This is not the case with internal transitions.

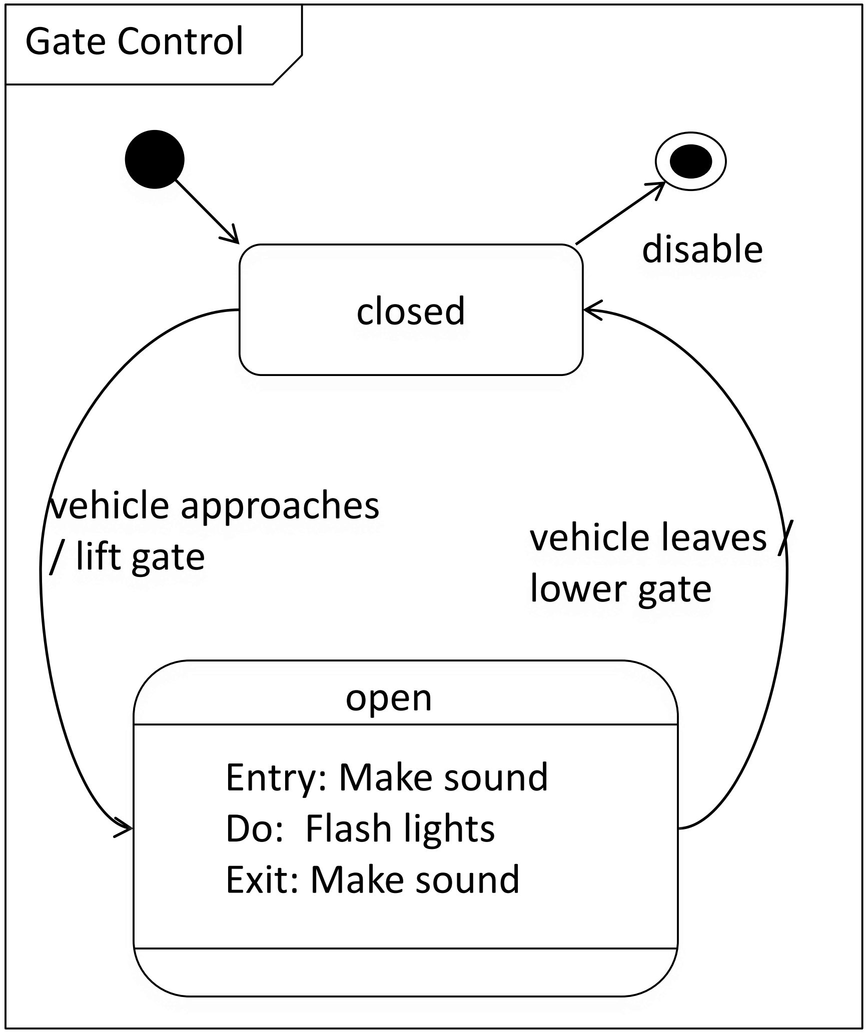


Recursive transition



Internal transition

Example: Gate Control revisited
Introducing state behavior

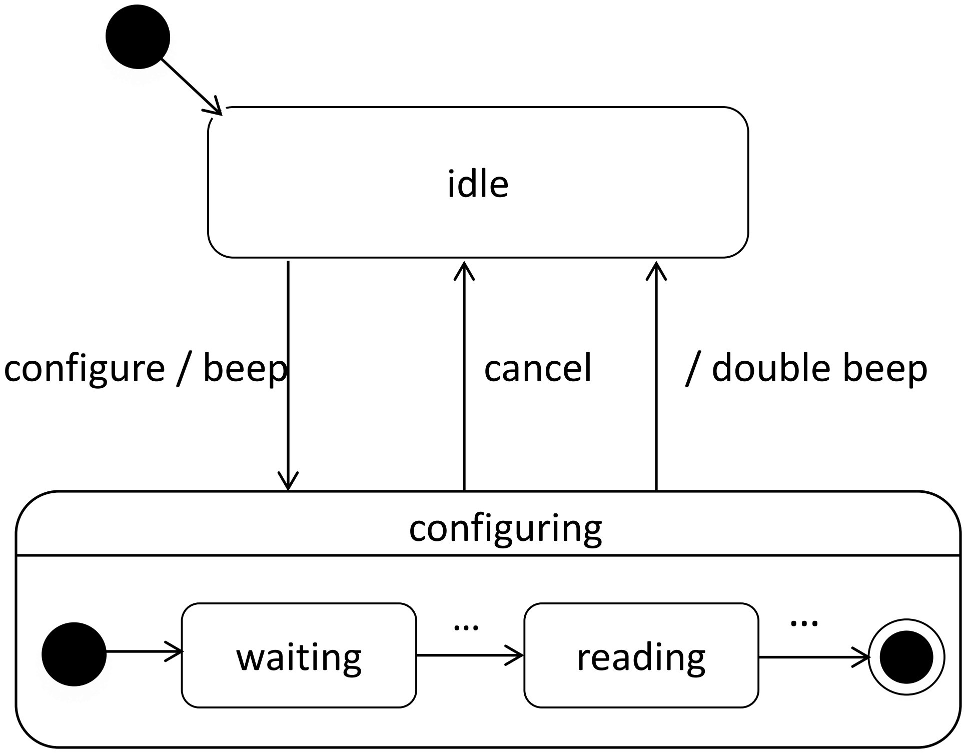


- In this extended example, the system will produce some sound upon entering and later upon exiting state open.
- While at state open, the system will flash some lights.

Hierarchically nested states

- A state (normally with complex behavior) can itself be modeled as a state machine.
- This implies that such a state will have its own tuple $(Q, \Sigma_1, \Sigma_2, q_0, V, \Lambda)$.
- Such states are called composite states, as opposed to simple states.
- States within a composite state are called nested states.
- The relation between a composite state and its nested states is that of a superstate and substate(s).

Hierarchically nested states

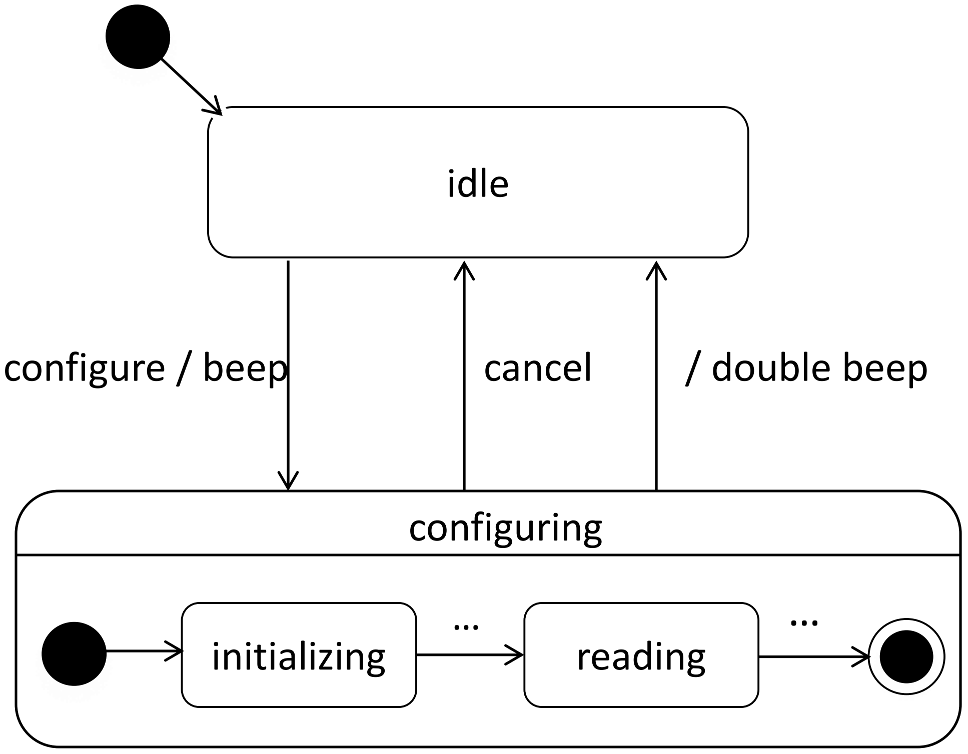


- State configuring is a composite state.
- It is modeled as a state machine.
- It has its own initial state, and its own exit state.
- Once configuring reaches its exit state, the enclosing region is completed and the system performs a transition to state idle.

Hierarchically nested states /cont.

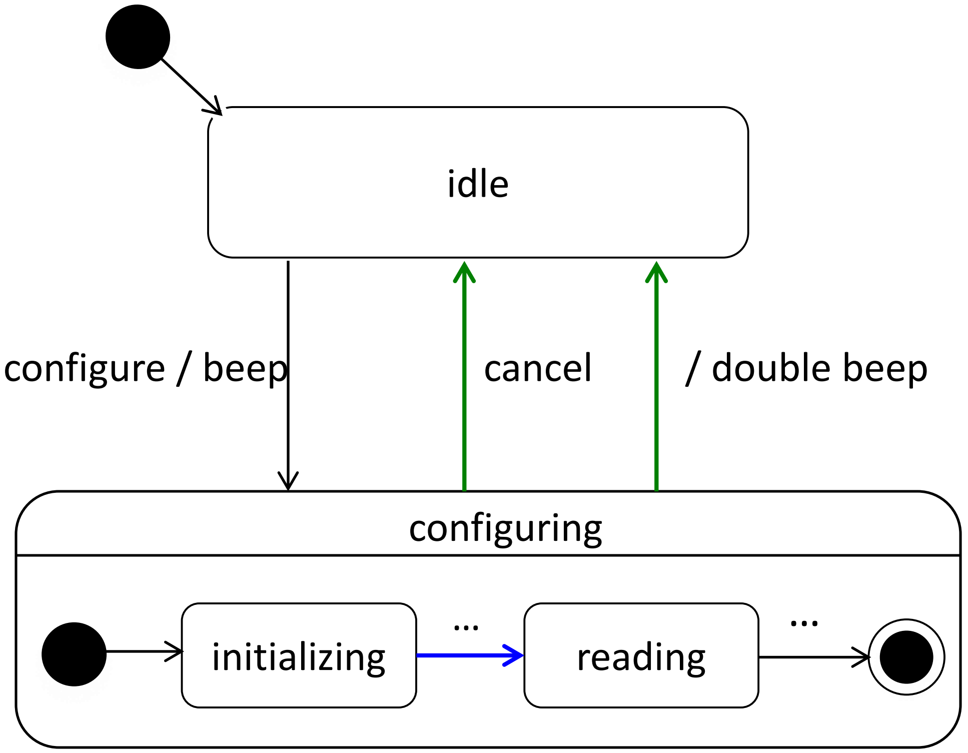
- A substate is called <u>direct substate</u>, as opposed to <u>transitively nested substate</u>, when it is not contained by any other state.
- If a system is in a substate, it is also (implicitly) in the superstate.
- Substates inherit the transitions of their superstate.

Hierarchically nested states /cont.



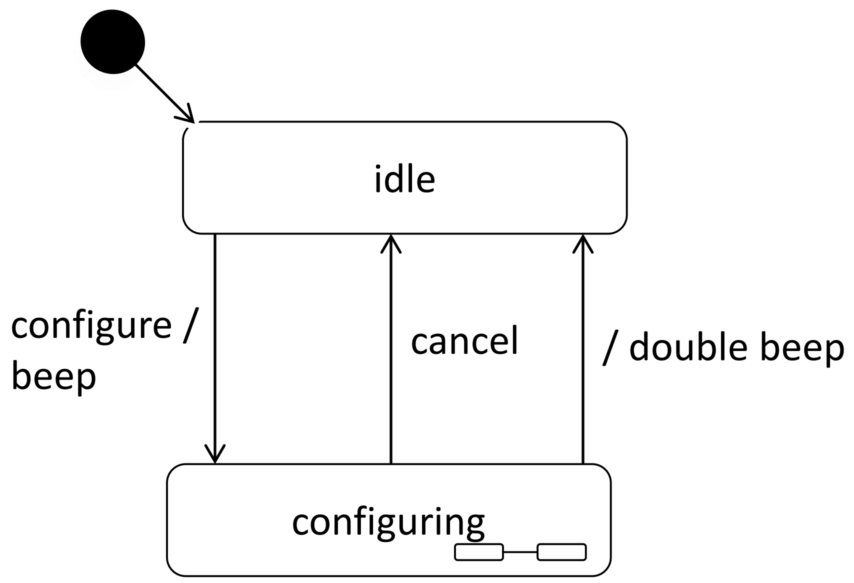
- States initializing and reading are nested.
- They are both the substates of configuring.
- Both initializing and reading inherit the transitions of configuring, their superstate.
- This means that while the system is at any of the two nested states, if the system receives event cancel, then it performs a transition to idle.

Internal and external transitions



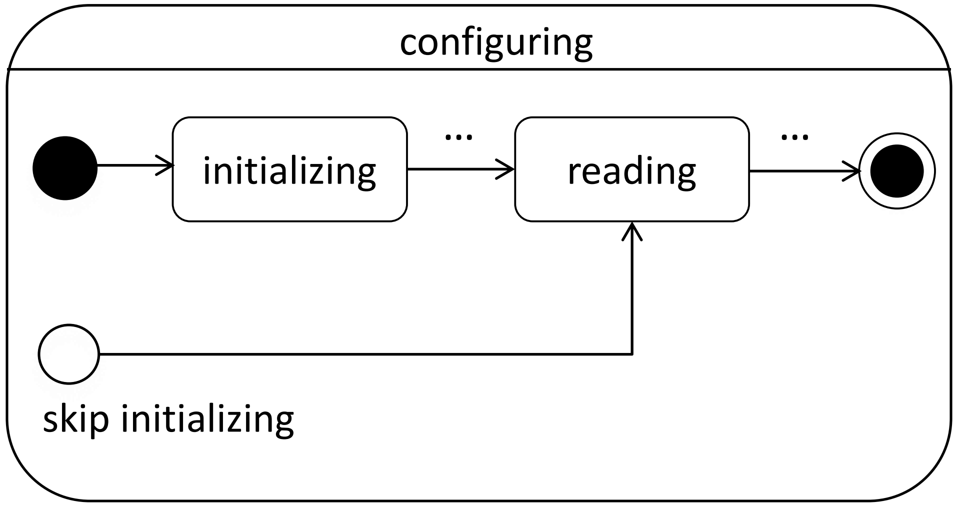
- Local transitions are inside a composite state.
- For example, the transition **from initializing to reading** is a local transition.
- External transitions leave a composite state.
- For example, the transitions **from configuring to idle** are external transitions.

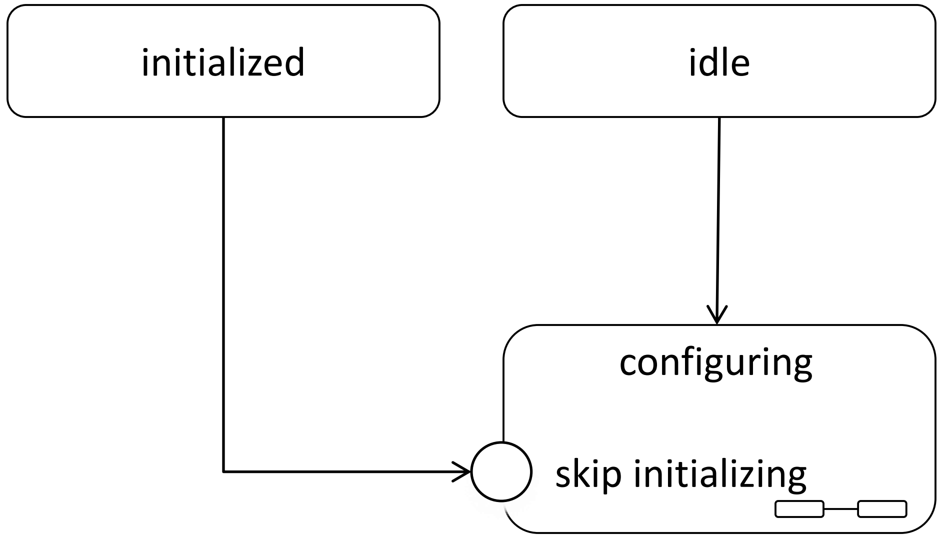
Abstracting composite behavior



- A composite state may be represented as a simple state, with an optional composite icon.
- This indicates that its decomposition is not shown in this particular diagram.

The Entry Point pseudostate



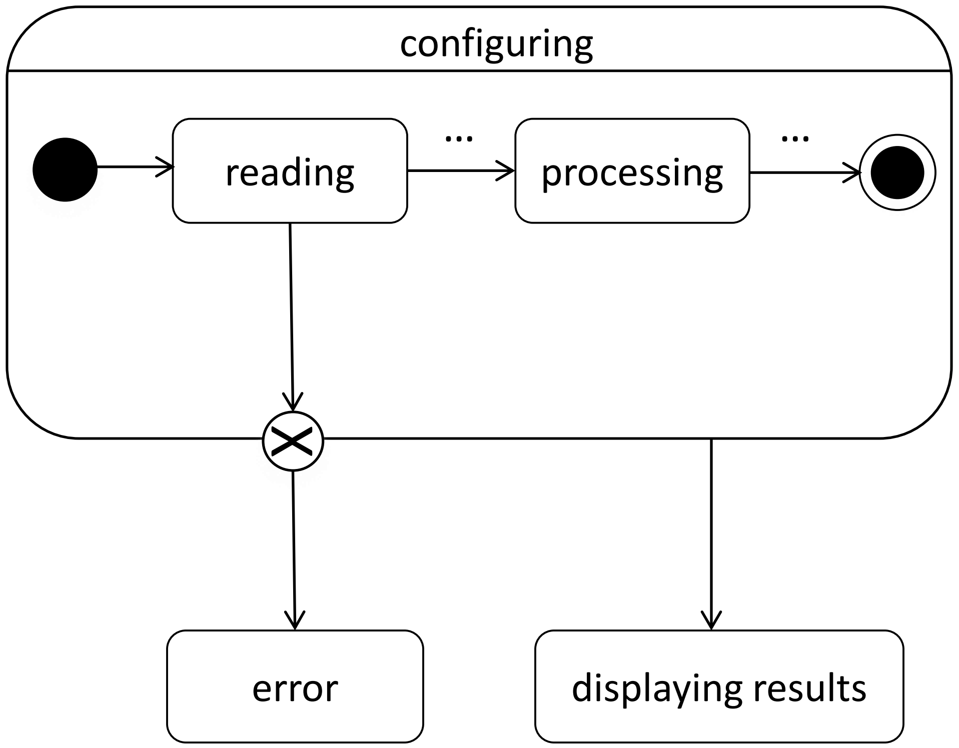


- The Entry Point pseudostate allows the system to enter a particular state in the composite state, other than its initial state.

- This is useful in conditions where re-entry into some composite state must not go through its initial state.

- We use the Entry Point symbol when we hide substate behavior.

The Exit Point pseudostate

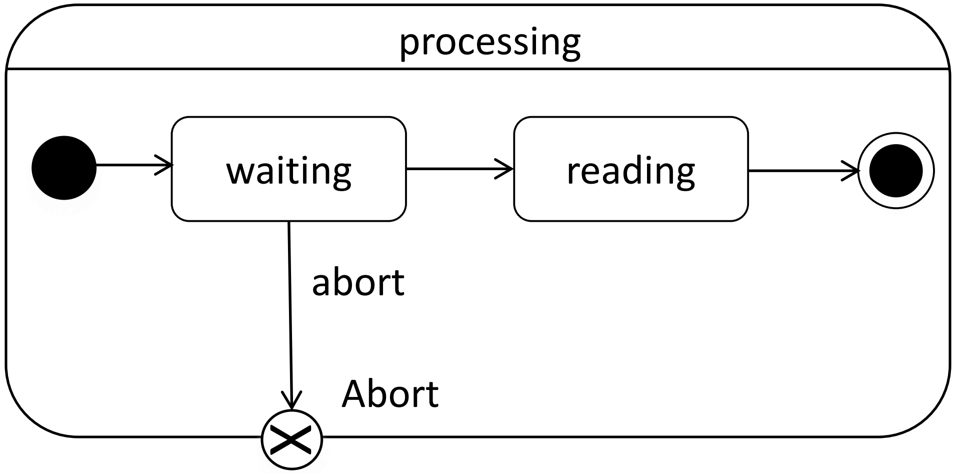


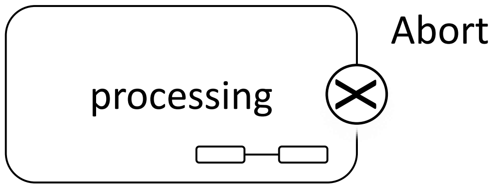
- The <u>Exit Point</u> pseudostate is an alternative exit point from a composite state.

- In the example, if the system fails to read, then it exits its superstate through the exit point, performing a transition to state error.

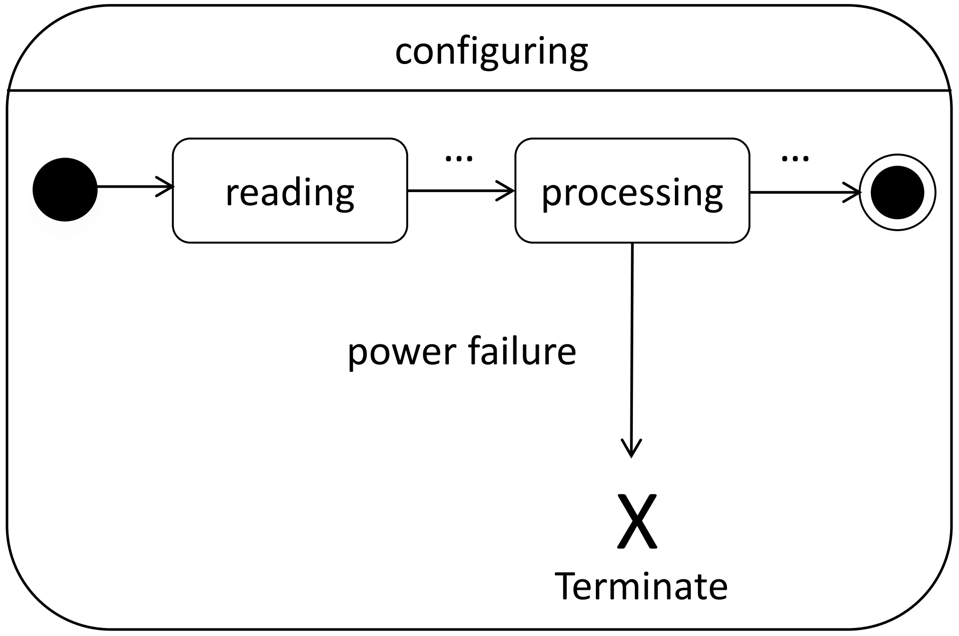
The Exit Point pseudostate /cont.

- The Exit Point pseudostate may optionally have a name.
- We use the Exit Point symbol when we hide substate behavior.





The Terminate pseudostate

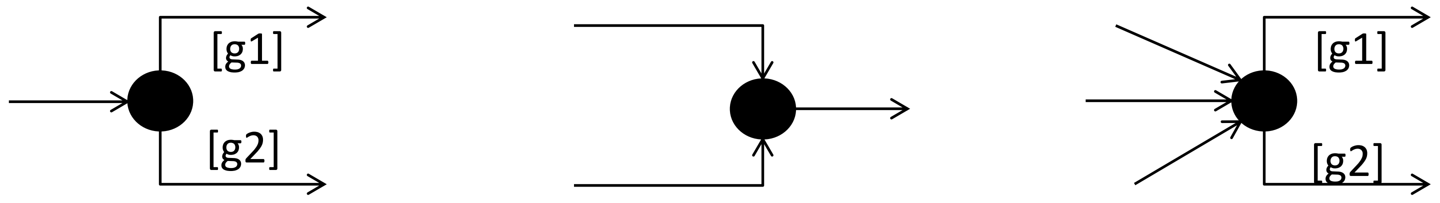


- The <u>Terminate</u> pseudostate ends the lifeline of the state machine.

- In the example, if there is a power failure while processing, then the function of the state machine ends.

The Junction pseudostate

- A junction pseudostate can be 1-to-many, many-to-1, or many-to-many:



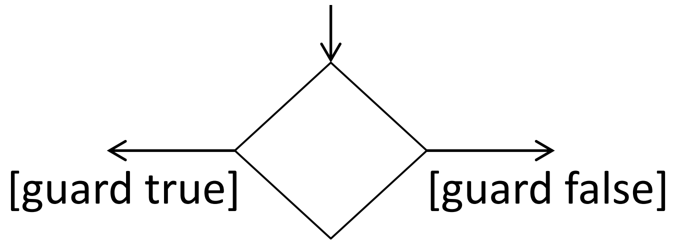
- 1-to-many: Used to split a single incoming transition into multiple outgoing transition segments with different guard conditions.

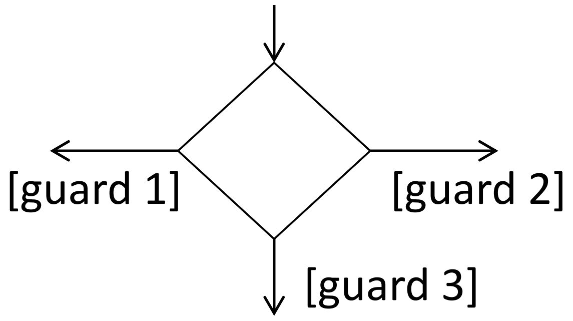
- Many-to-1: Used to converge (or merge) multiple incoming transitions into a single outgoing transition representing a shared transition path.

- Many-to-many: Multiple incoming and multiple outgoing transitions segments with different guard conditions.

The Choice pseudostate

- The Choice pseudostate receives a single incoming transition and produces two (or more) transitions, each with a guard condition, one of which is true.

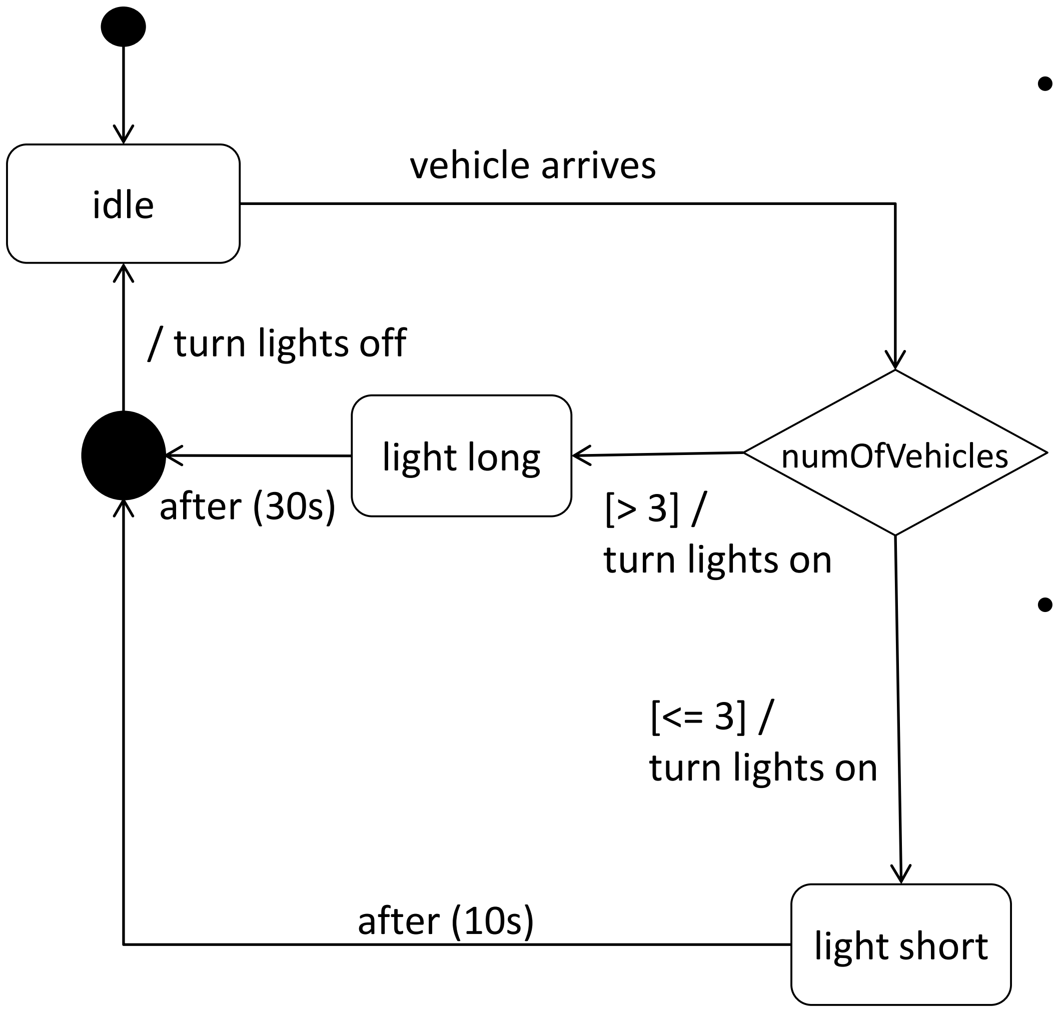




- If all guards share a left operand, the operand can be placed inside the diamond.
- If none of the guards evaluates to true, then the model is considered badly-formed.
- To avoid this, we should define one outgoing transition with the predefined `else` guard when appropriate.

Example: Vehicle passageway

Introducing junction and choice pseudostates

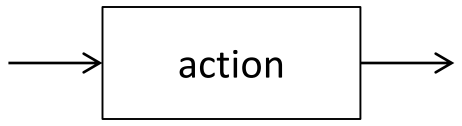


- If while idle, a vehicle arrives, then the system will perform a transition to either light long or light short depending on the evaluation of the guard expressions.
- Notice that the guard expressions are mutually exclusive.

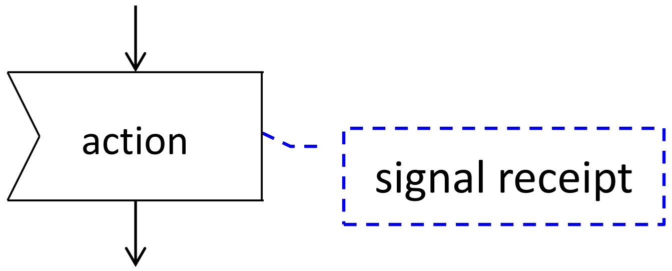
Actions revisited

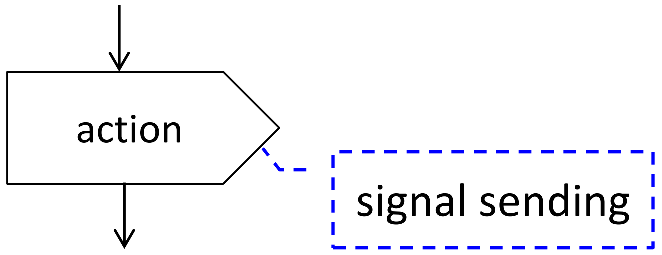
- We have seen that actions can be denoted by a slash over a transition as part of the tripple `event [guard] / action`

- Actions can also be denoted within a rectangle.

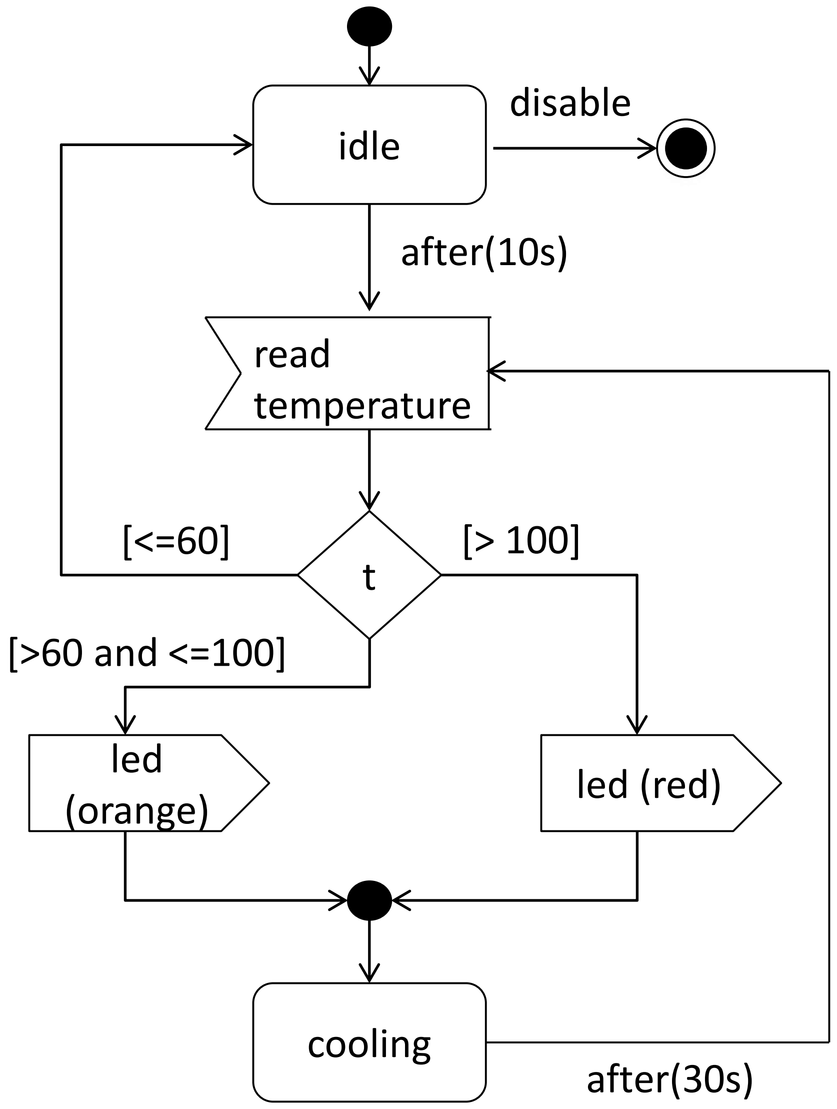


- Actions that send or receive signals can be represented by appropriate symbols:



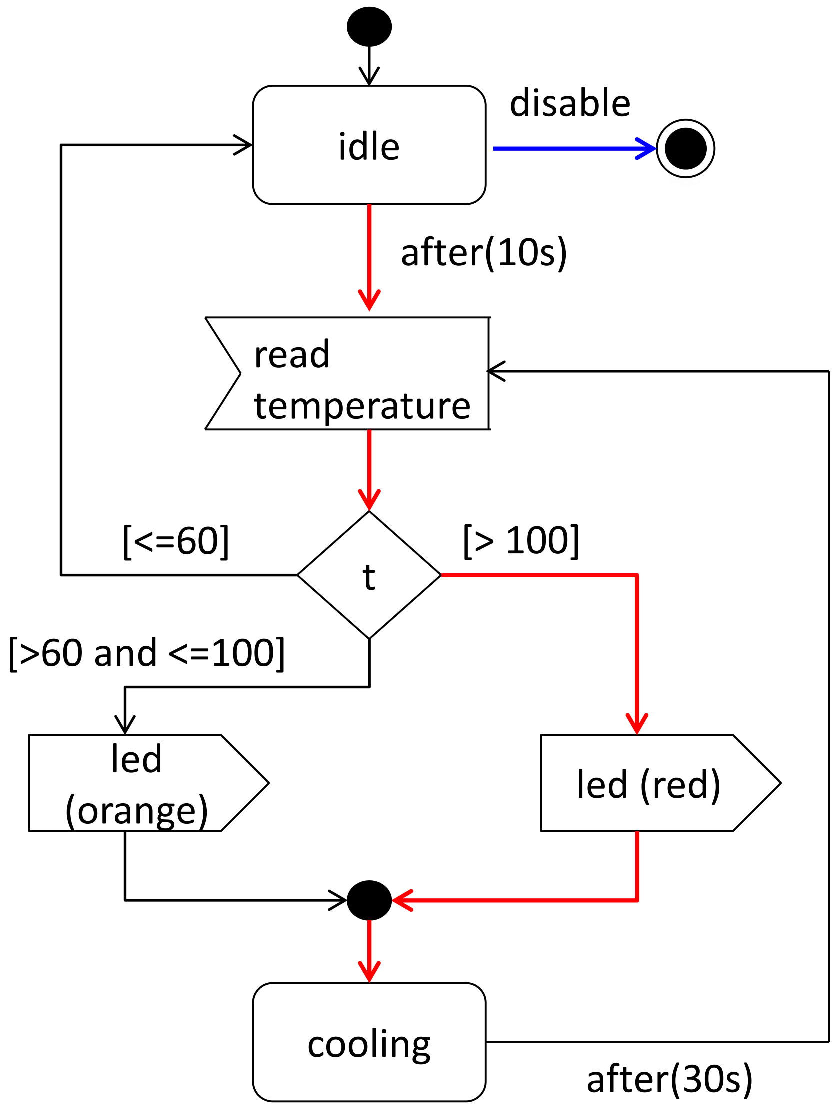


Example: A cooling system



- Once idle, the system reads the temperature after some time.
- It will perform a transition to cooling or back to being idle.
- During the transition to cooling, the system performs different actions based on the value of each guard.

Simple vs. Compound transitions



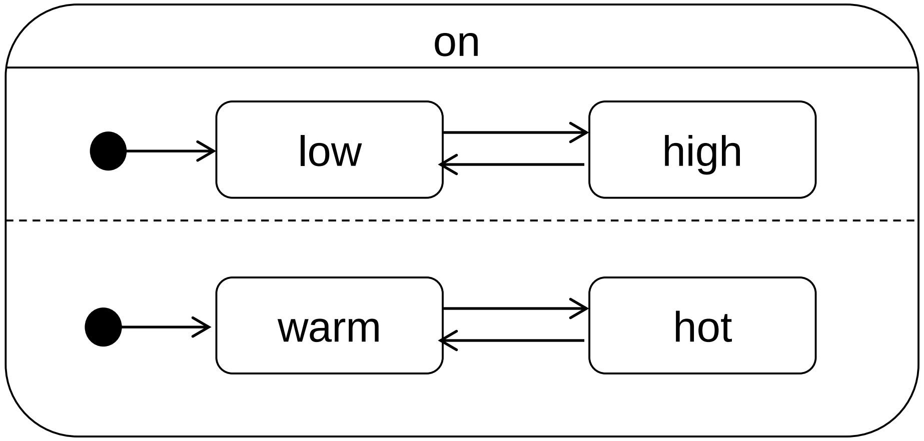
- A simple transition connects two states directly.
- A compound transition consists of several transitions that use one or more pseudostates to link one state with another.
- The transition from idle to exit is a simple transition.
- The transition from idle to cooling is an example of a compound transition.

Orthogonal states and concurrent regions

- A state may be divided into concurrent (or parallel) regions.

- Such a state is called orthogonal.

- The concurrent regions within an orthogonal state operate simultaneously and independently.

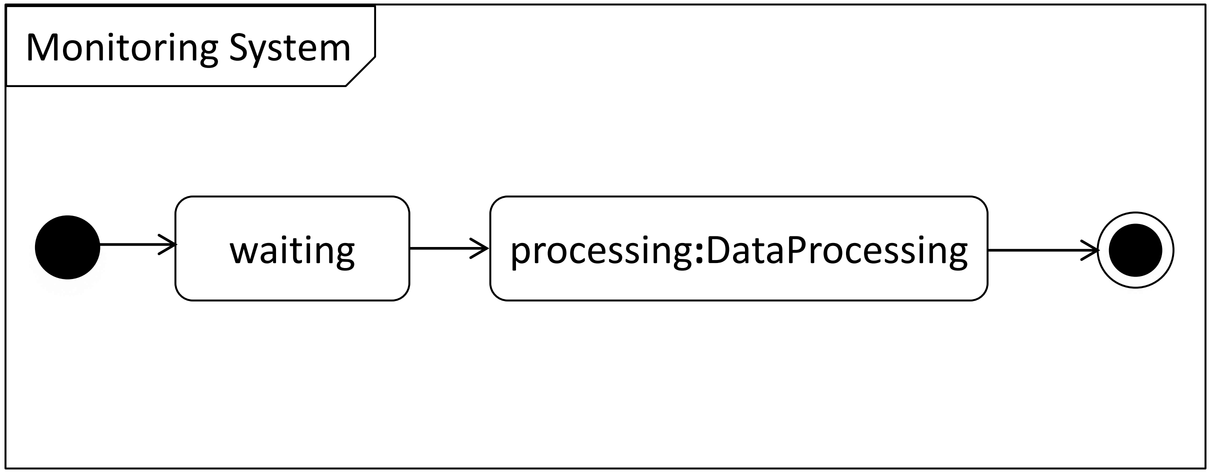


- The system may be in any of (on, low, warm), (on, low, hot), (on, high, warm), or (on, high, hot).

Submachine states

- A submachine state references an existing state machine.
- An existing state machine is referenced by a state as

  <name of state> **:** *<name of existing state machine>*



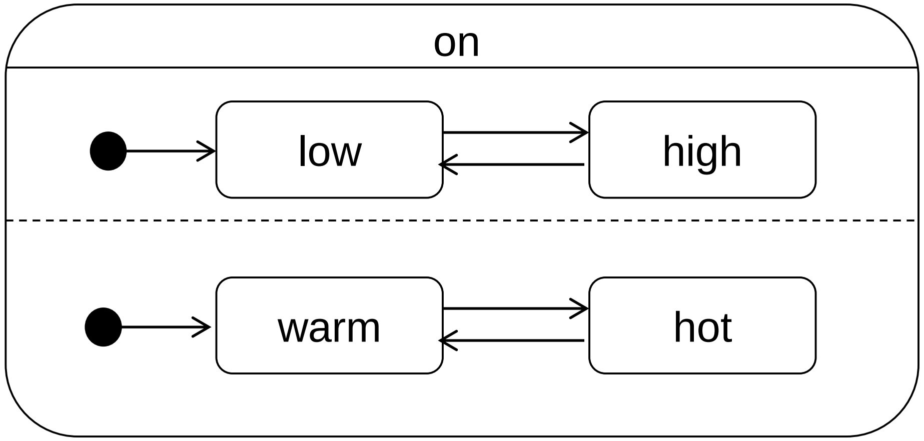
States revisited: Types of states

- There are 4 types of states:

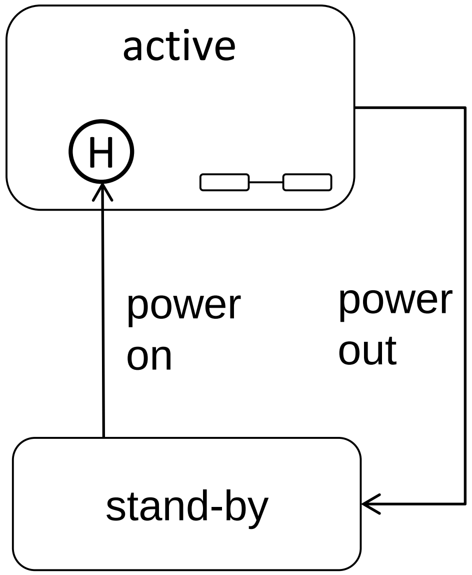
1. A <u>simple</u> state contains no region.
2. A <u>composite</u> state contains at least one region.
3. An <u>orthogonal</u> state contains at least two regions.
4. A <u>submachine</u> state references a statemachine.

Example: Shallow and deep history

- Without history, as soon as the heater is switched on, it jumps into the default state configurations low and warm.
- Let us now switch the heater to high and hot.
- Next time we use the heater, it will start at low and warm.
- As is, the system maintains no history i.e. it has no memory of its last active configuration.



Shallow History pseudostate

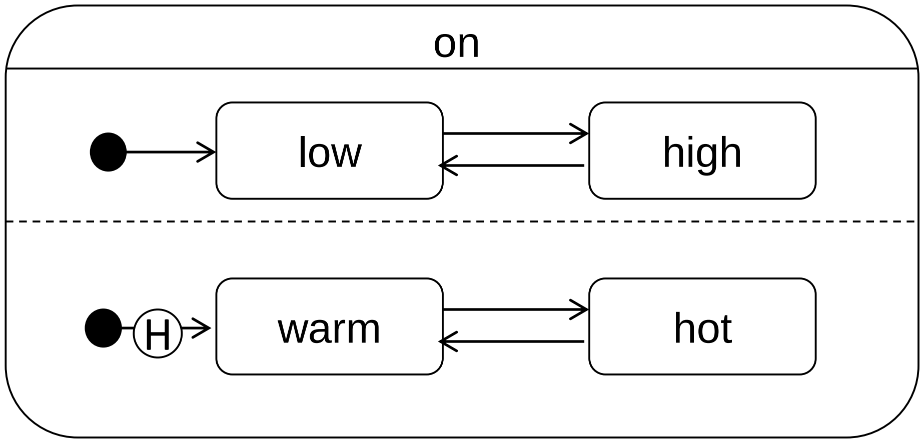


- The Shallow History pseudostate causes the last active substate of a region to be stored.
- When traversing into the region, the last active substate will be automatically activated as if it were the initial state.

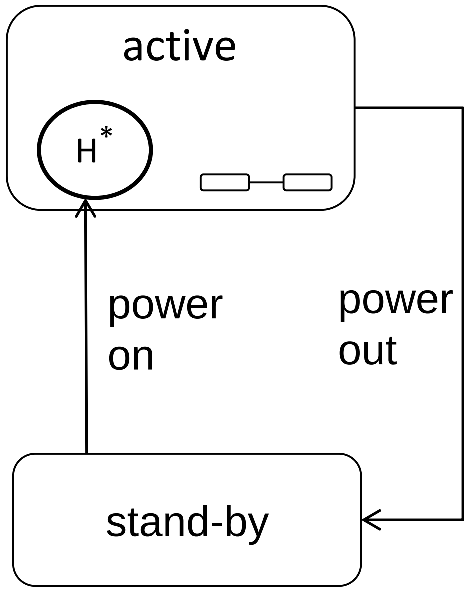
Example: Shallow and deep history /cont.

- Next time the system performs a transition into state on, it will remember the last active configuration of the region with the Shallow History.

- As a result, it will immediately go to low (initial state of upper region) and hot (last active state in the lower region).



Deep History pseudostate

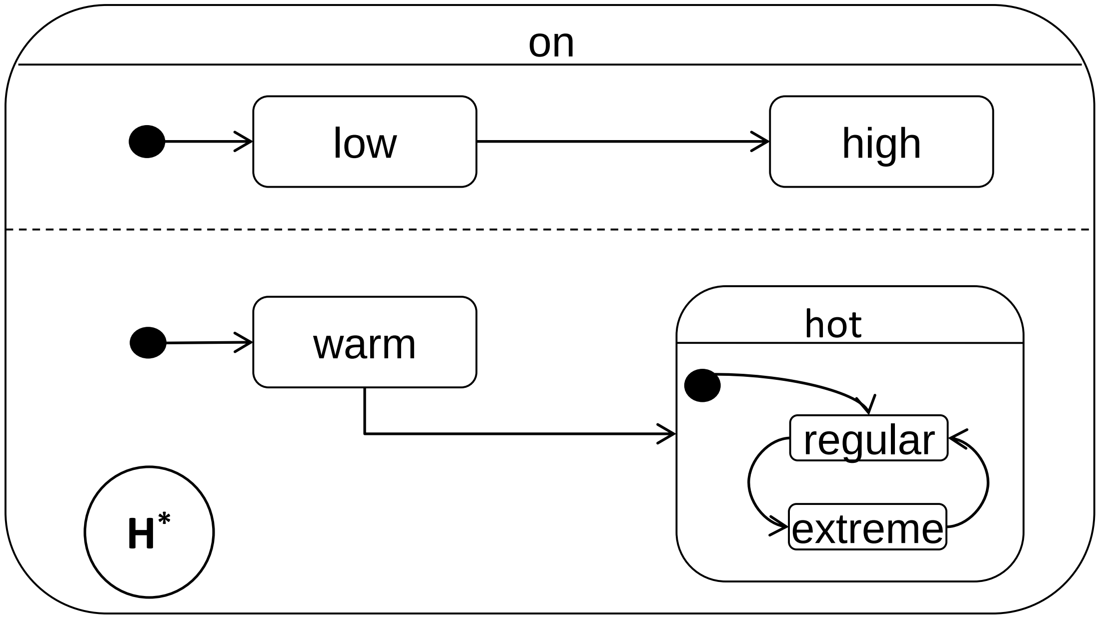


- The Deep History pseudostate maintains a memory of inner regions of the substate.

Example: Shallow and deep history /cont.

- Let us extend state hot to include states regular and extreme, and switch the heater to extreme.

- Next time the system performs a transition into state on, it will immediately jump into low and extreme.

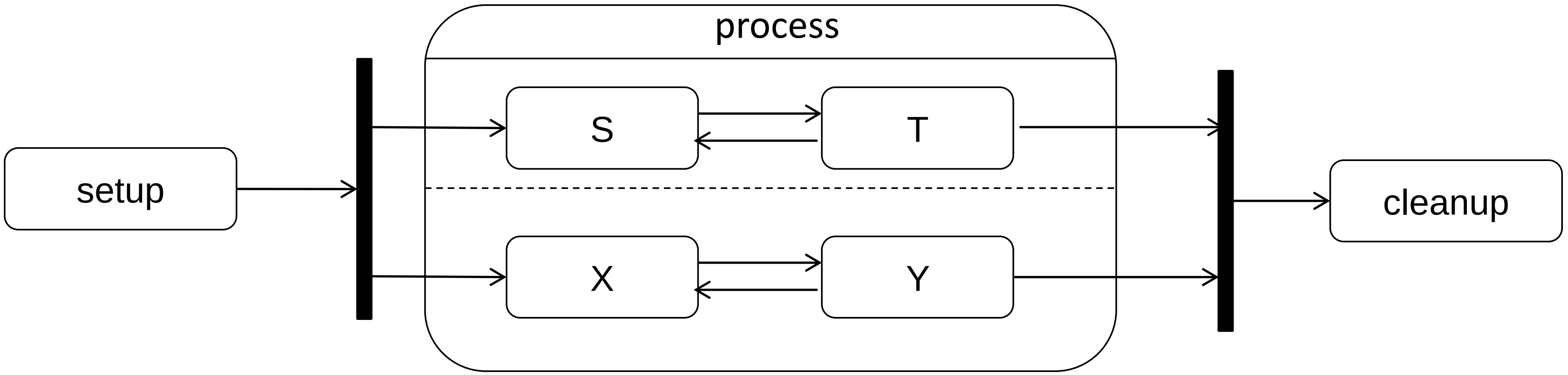


The Fork and Join pseudostates

- The Fork pseudostate receives one incoming transition that splits into two or more outgoing transitions.

- The outgoing transitions lead to different concurrent regions.

- Outgoing transitions must have no guards and no actions.



- The Join pseudostate receives two or more incoming transitions from concurrent regions that meet to form one outgoing transition.

PART 2: Protocol State Machines

Protocol state machines

- A protocol state machine is a specialization of a behavioral state machine.
- It focuses on the sequences of events a component responds to (called event protocols), without showing its behavior.

Protocol state machines /cont.

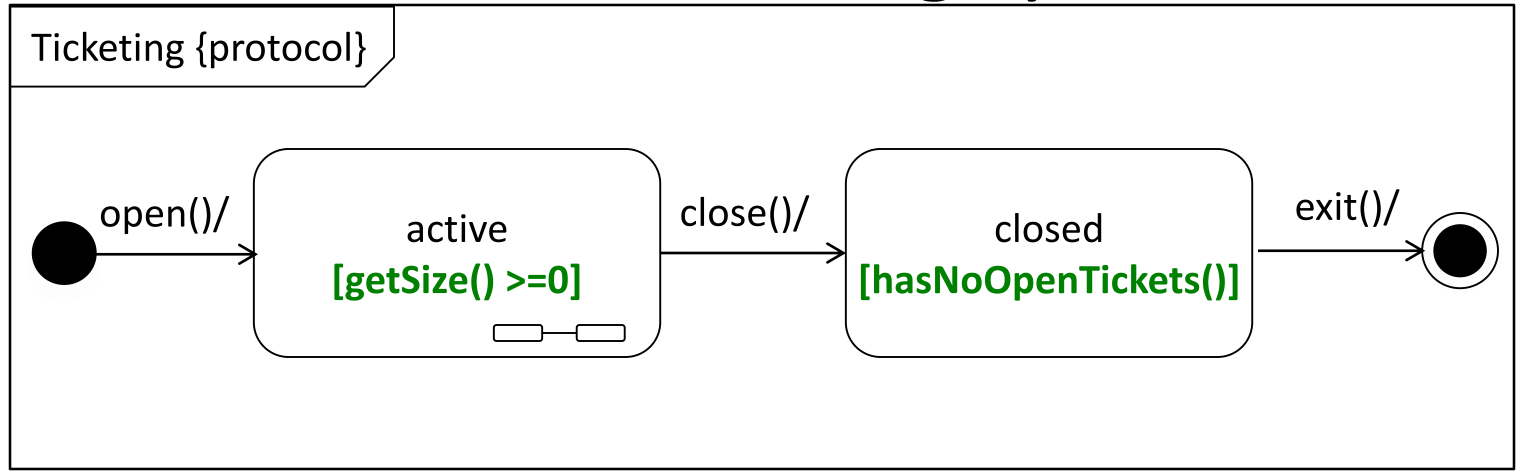
- In what ways does a protocol state machine differ from a behavioral state machine?

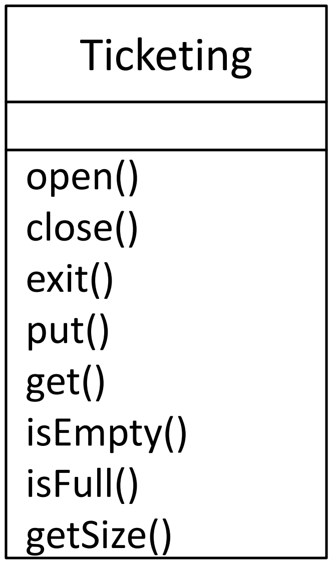
- States:
  - May contain an invariant expression.
  - Do not show entry/do/exit activities.

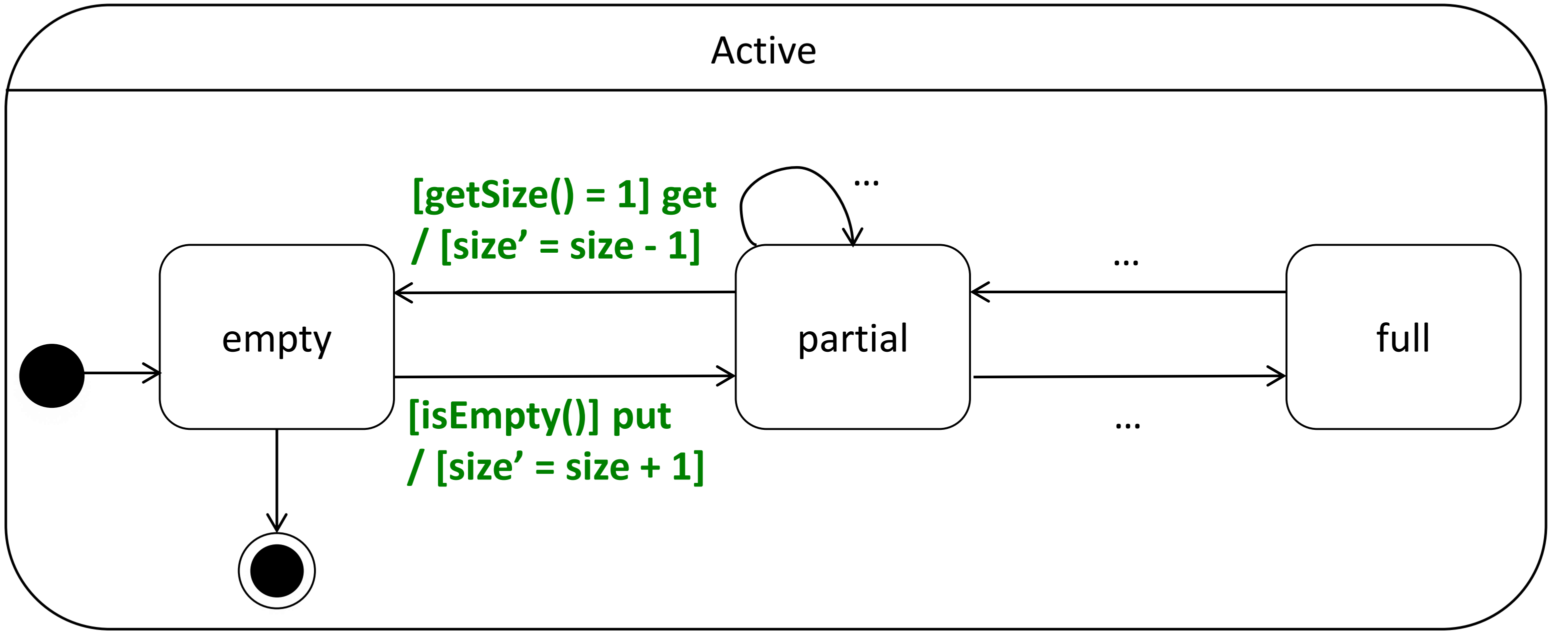
- Transitions
  - Do not show actions.
  - Can be associated with assertions (preconditions and postconditions):

    [precondition] event / [postcondition]

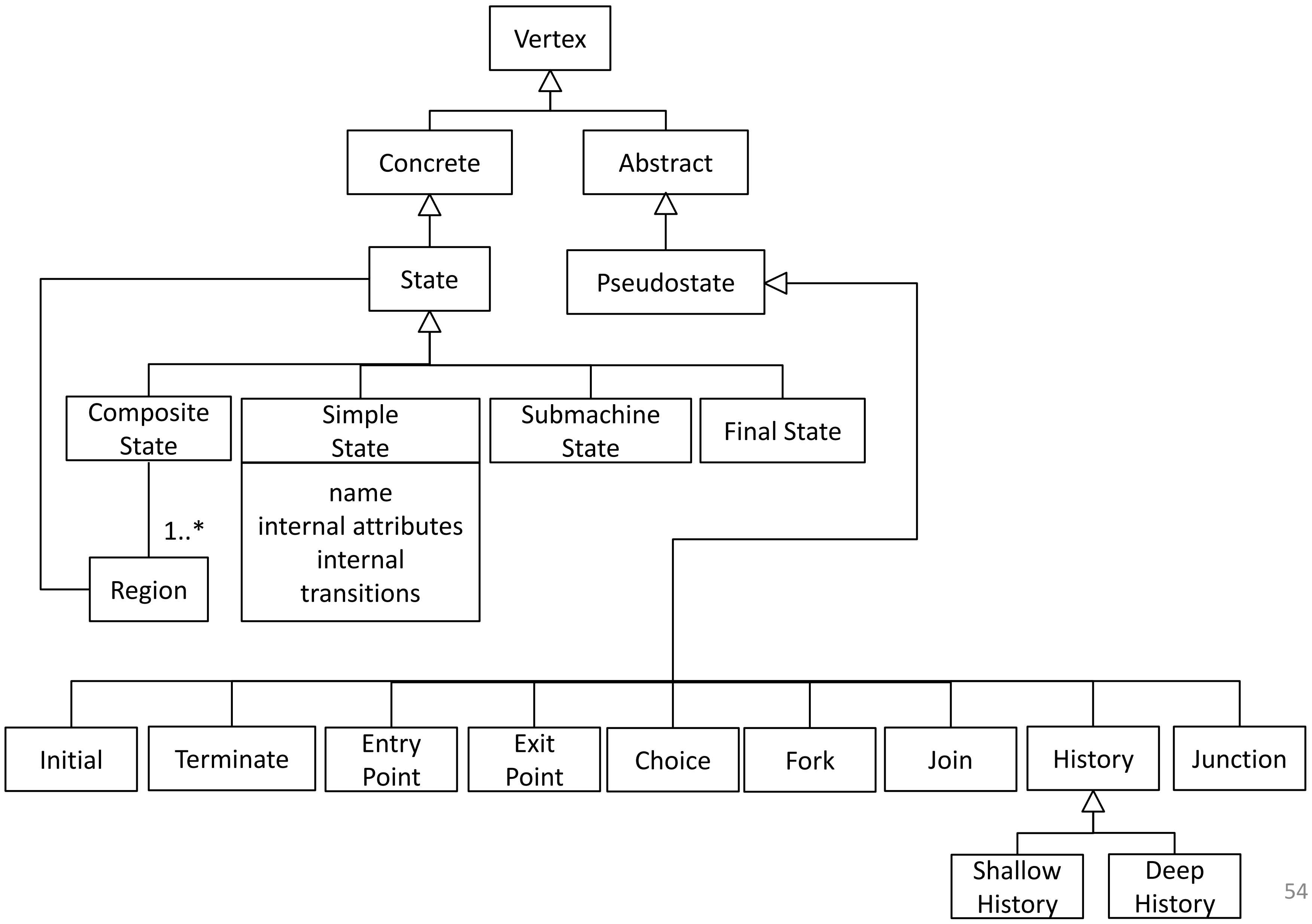
Example: A protocol state machine for a ticketing system







State machine metamodel


Vertex
Concrete
Abstract
State
Pseudostate
Composite State
Simple State
name
internal attributes
internal transitions
Submachine State
Final State
1..*
Region
Initial
Terminate
Entry Point
Exit Point
Choice
Fork
Join
History
Junction
Shallow History
Deep History