Stateflow Tutorial

Lecture 61, v01

John G. Artus

BSEE

MSSE

INCOSE ESEP

About Sources of Material

- Much of the source material used to generate this lecture comes from external sources that are identified in the References Section
- In this lecture, the vast majority of the material comes from The Mathworks, the producer of the software tool Stateflow, which is the subject of this lecture
- Stateflow provides a graphical language that models state transition diagrams, flow charts, state transition tables, and truth tables to describe how a system reacts to input signals, events, and time-based conditions
- I have used the source material to generate a lecture flow which is best suited for a university educational setting
- My copyright addresses the structuring of this source material in a form which best suits this educational objective
- I make no claim of ownership or authorship of material obtained from sources identified in the References Section
  - John G. Artus

About this Document

- The purpose of this Lecture is to provide students with Start-Up Guidance that cen help get them up and going with Simulink/Stateflow without the need to refer to text documentation hundres of pages long
- However, the material in this Lecture only covers a small fraction of the thousands of syntax rules encompassed by the Stateflow modeling tool
- To locate additional information beyond what is provided here, it is recommended that the student pursue the following Stateflow official documentation from The MathWorks website (addresses are provided in the References Section)
  - Stateflow Guide To Getting Started
  - Stateflow User's Guide

Sections

Section 1

The MathWorks Products

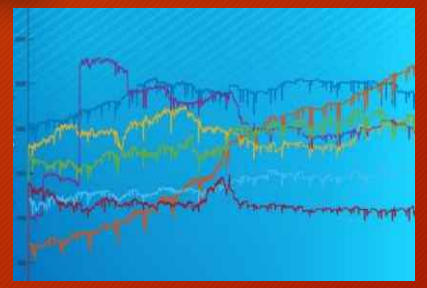
- MATLAB
- Simulink
- Stateflow

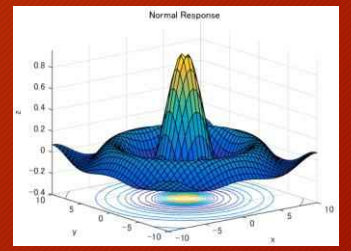
Objective

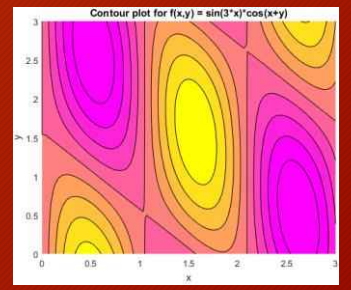
- Use Stateflow to model and understand system states and the events and conditions that trigger transitions between those states
- MATLAB-Simulink-Stateflow
  - MATLAB
    - Used to perform iterative analysis and design with a programming language that expresses matrix and array mathematics directly
  - Simulink
    - Used to develop graphical models using drag-and-drop of modeling components from an extensive library to simulate and test the system early and often
  - Stateflow
    - Provides a graphical language that models state transition diagrams, flow charts, state transition tables, and truth tables to describe how a system reacts to input signals, events, and time-based conditions

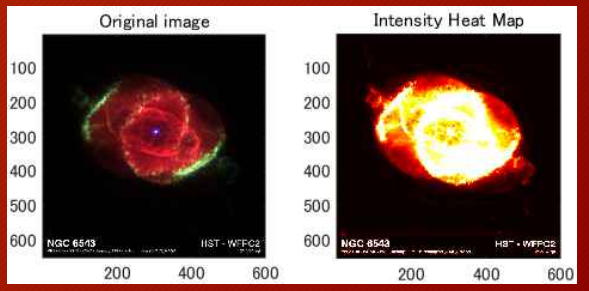
MATLAB

- MATLAB combines a desktop environment tuned for iterative analysis and design processes with a programming language that expresses matrix and array mathematics directly
- With MATLAB, you can create scripts that combine code, output, and formatted text in an executable notebook
- MATLAB apps let you see how different algorithms work with your data
- Iterate the problem solution until you've got the results you want
  - Then generate a MATLAB program to reproduce or automate your work
- Organize, clean, and analyze complex data sets
- Create visualizations from built-in libraries
  - Use built-in plots to visualize your data, gain insights, and identify underlying patterns and trends
  - Explore function syntax and available chart options using the integrated documentation
  - Choose from the relevant plots presented, based on data you've selected
  - This lets you find the optimal visualization for your data
- MATLAB provides the tools you need to transform your ideas into algorithms
  - Thousands of core mathematical, engineering, and scientific functions
  - Application-specific algorithms in domains such as signal and image processing, control design, computational finance, and computational biology
  - Development tools for editing, debugging, and optimizing algorithms



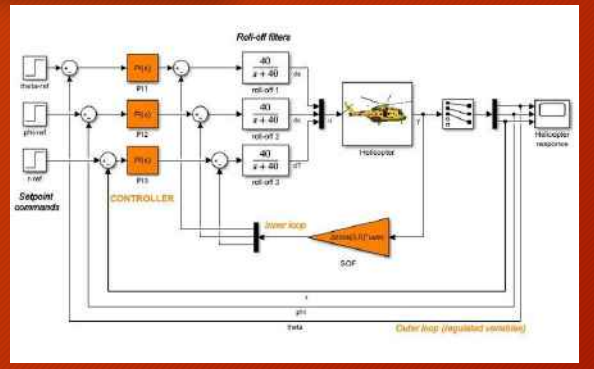


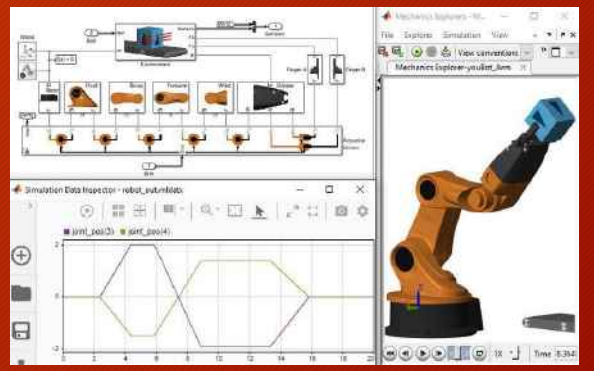


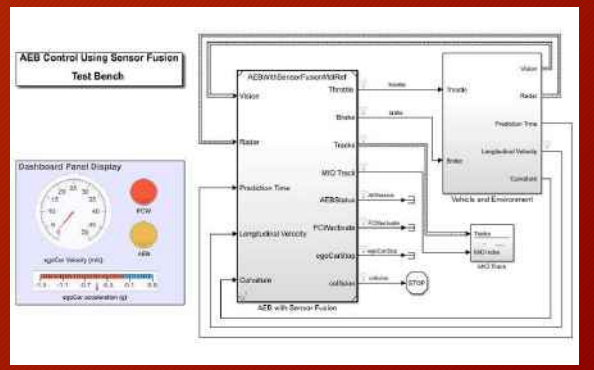


Simulink

- Use models systematically throughout the entire system developments process
- Use a virtual model to simulate and test the system early and often
- Validate the design with physical models, Hardware-in-the-Loop testing, and rapid prototyping
- Generate production-quality code and deploy directly to the embedded system
- Maintain a digital thread with traceability through requirements, system architecture, component design, code and tests
- Extend models to systems in operation to perform predictive maintenance and fault analysis

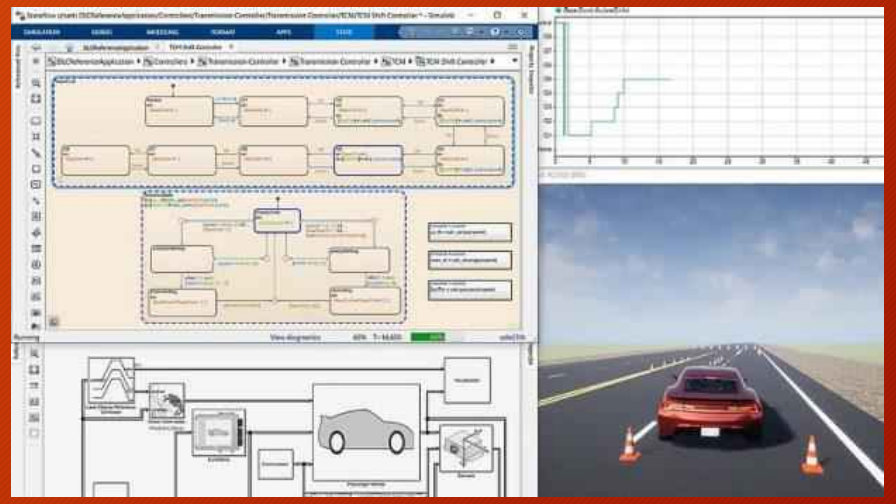


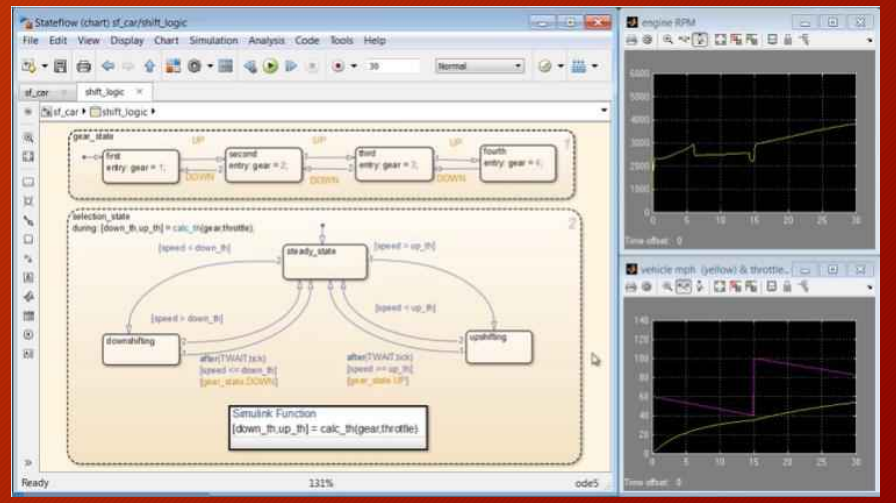




Stateflow

- Stateflow provides a graphical language that includes state transition diagrams, flow charts, state transition tables, and truth tables
- You can use Stateflow to describe how MATLAB algorithms and Simulink models react to input signals, events, and time-based conditions
- Stateflow enables you to design and develop supervisory control, task scheduling, fault management, communication protocols, user interfaces, and hybrid systems
- With Stateflow, you model combinatorial and sequential decision logic that can be simulated as a block within a Simulink model or executed as an object in MATLAB
- Graphical animation enables you to analyze and debug your logic while it is executing
- Edit-time and run-time checks ensure design consistency and completeness before implementation





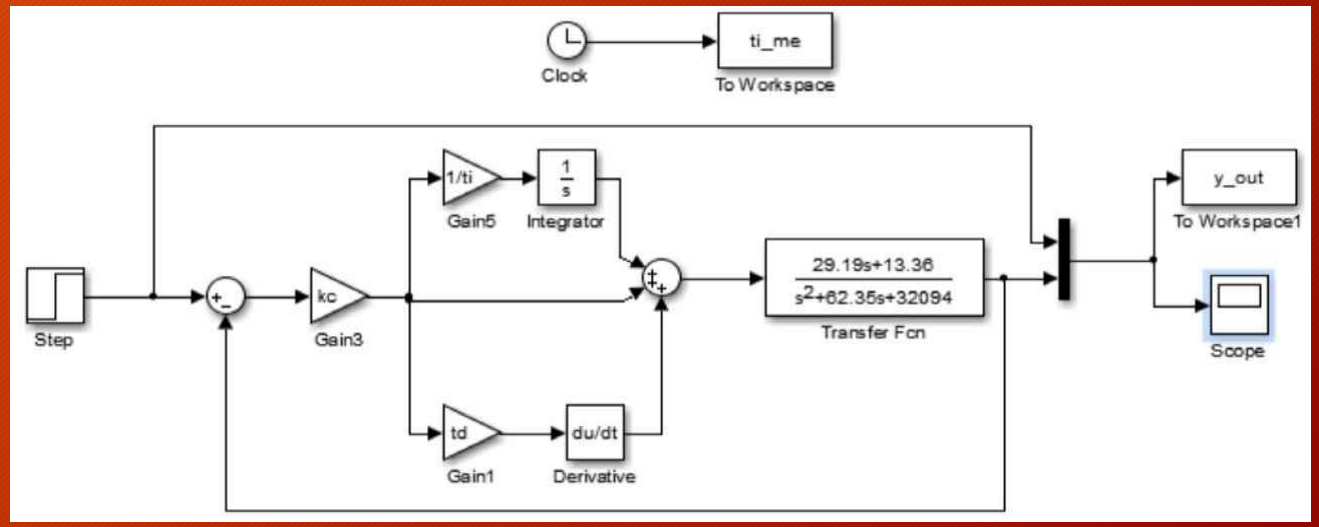
Section 2

Connection between Simulink and Stateflow

Simulink Modeling

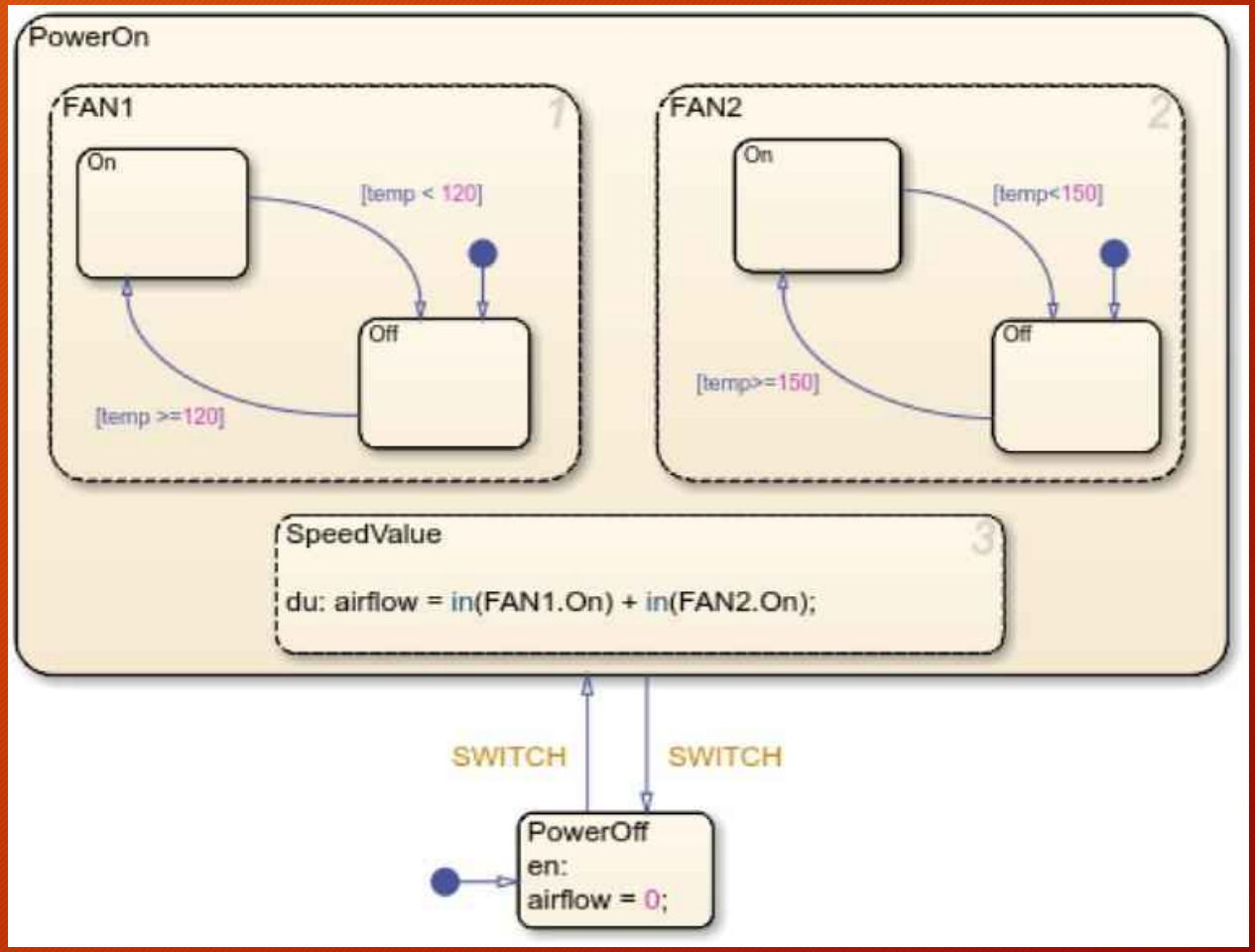
- Simulink is a MATLAB-based graphical programming environment for modeling, simulating, and analyzing multidomain dynamic systems
- Its primary interface is a graphical block diagramming tool and a customizable set of block libraries
- It offers tight integration with the rest of the MATLAB environment and can either drive MATLAB or be scripted from it
- Simulink is widely used in automatic control and digital signal processing for multidomain simulation and model-based design

Model of the Performance Index for an Integral of absolute error (IAE) for a Synchronization Control Scheme for a Triaxial Motion System



Stateflow Modeling

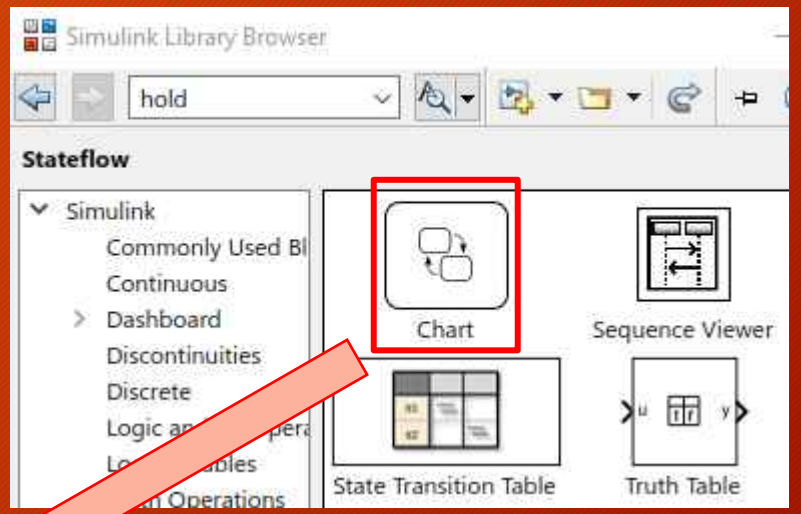
- Stateflow is a control logic tool used to model reactive systems via state machines and flow charts within a Simulink model
- Stateflow uses a variant of the finite-state machine notation enabling the representation of hierarchy, parallelism, and history within a state chart
- Stateflow also provides state transition tables and truth tables

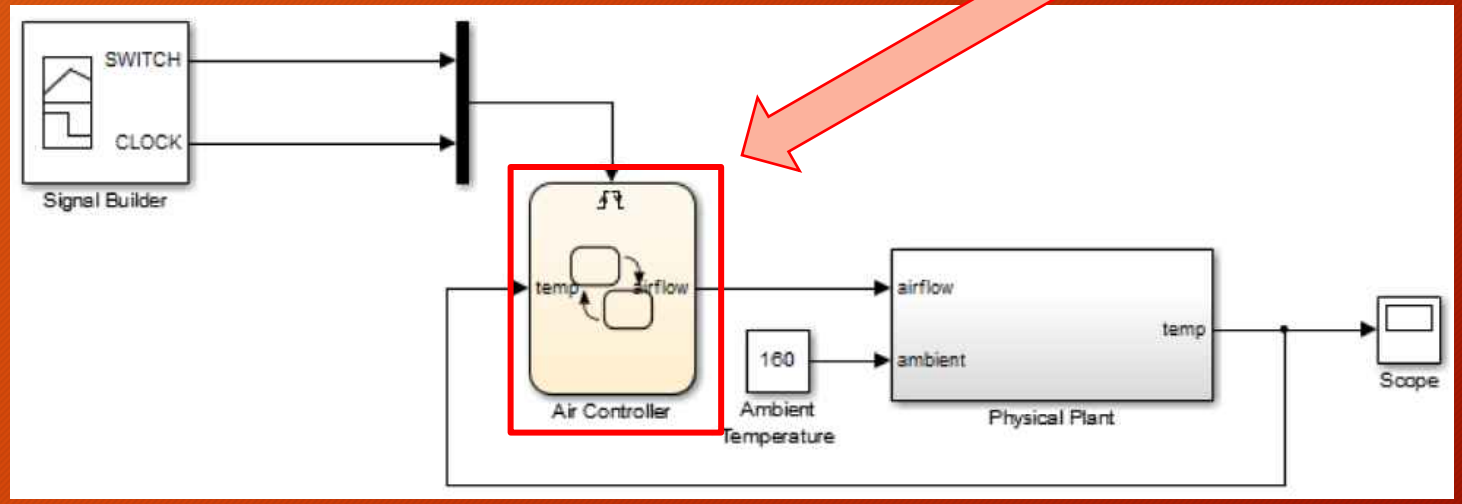


Stateflow model of a an air controller that maintains air temperature in a physical plant

Integrating Stateflow with Simulink

- To construct a Stateflow chart
  - Add a Stateflow Chart block to the Simulink model
- To integrate the Stateflow chart with Simulink
  - Add inputs and outputs to the Chart block by
    - Double-click the Chart block to enter the Statechart
    - In the Modeling tab, under Design Data, select Data Input
    - Fill in the Data properties dialog box





Stateflow Chart Masks

- You can creating masks for
  - Stateflow charts
  - State transition tables
  - Truth tables
- Creating masks for Stateflow simplifies how you use and share blocks in a Simulink model
  - It encapsulates the block by hiding the underlying logic
  - It creates a user interface for the block
- You can then customize the Chart block by:
  - Changing the appearance with meaningful icons and ports
  - Creating a user interface for parameters
  - Adding customized documentation
- You decide which parameters to change through the mask user interface
  - You can provide meaningful descriptions of these parameters
- To open the Mask Parameters dialog box
  - Double-click the Stateflow chart
  - A dialog box opens which contains the set of existing parameters that can be modified, added to, or deleted

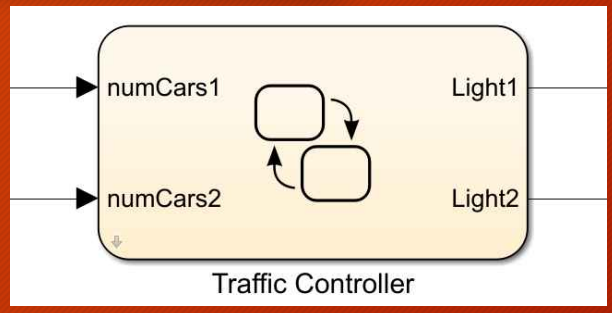
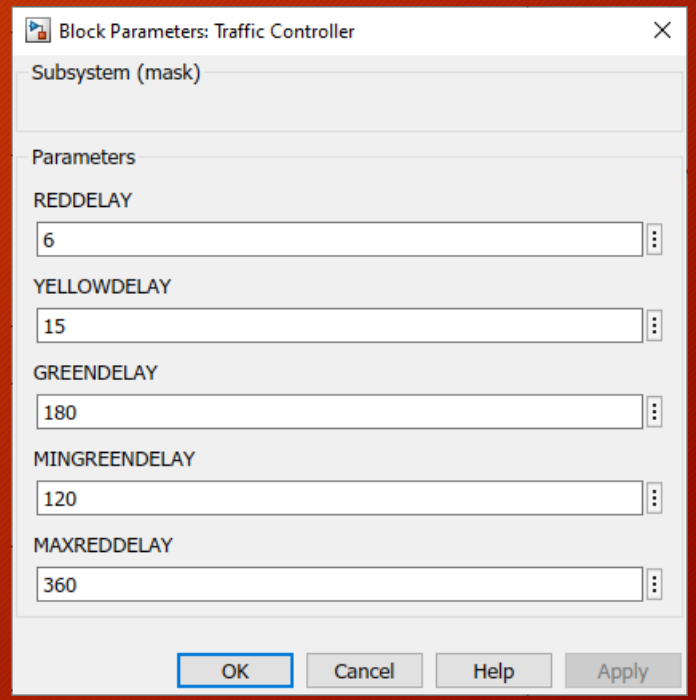


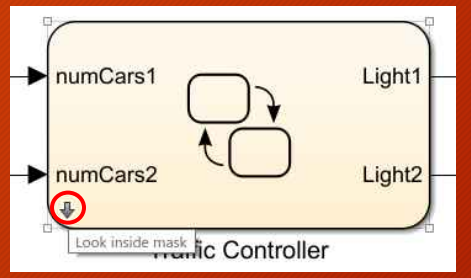
Chart Mask for Trafic Light Model



Parameter list for Trafic Light Model Chart Mask

Under the Mask

- Look Under the Mask
  - You can view and edit the contents of a masked block by clicking the Look Inside Mask “badge” on the chart
  - The badge is a downward facing arrow in the lower-left corner of the chart
  - Looking under a mask does not unmask the block



- Edit the Mask
  - To edit a mask, in the State Chart tab, click Edit Mask
  - In the Mask Editor, you can modify the mask icon, change the parameters, or add documentation
  - To remove the mask, click Unmask in the lower corner of the Mask Editor
  - After you change a mask, click Apply to save the changes

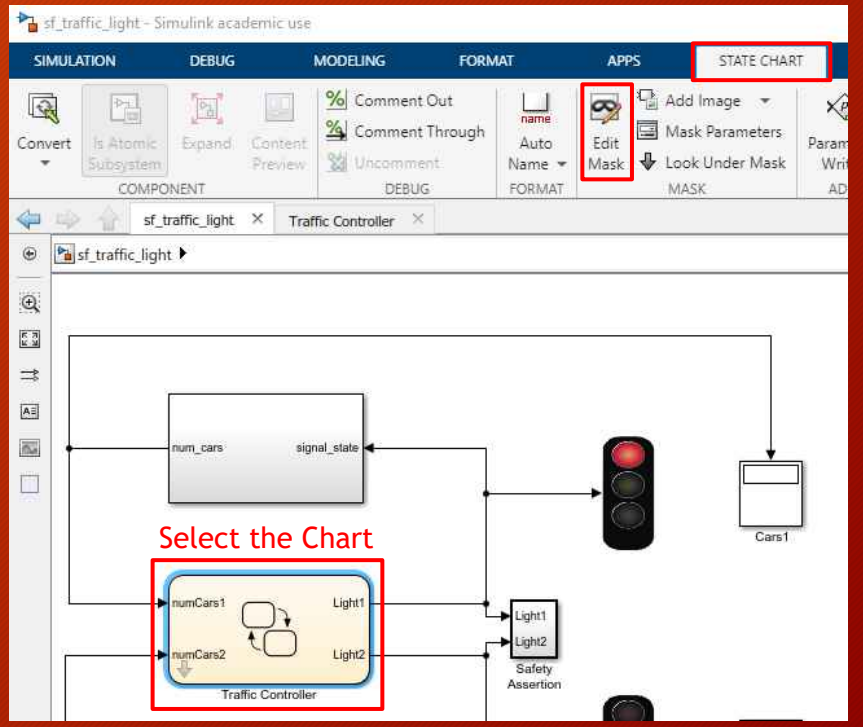


Chart Inputs and Outputs

- The Simulink model passes data inputs to the Stateflow chart and receives outputs from the Stateflow chart
- Inputs and outputs are data elements in a Stateflow chart that interact with the parent Simulink model
- For standard, simple inputs and outputs for your chart, follow these steps:
  1. Double-click the Chart block in the Simulink model to open the Stateflow chart

     The Stateflow Editor opens on your desktop:

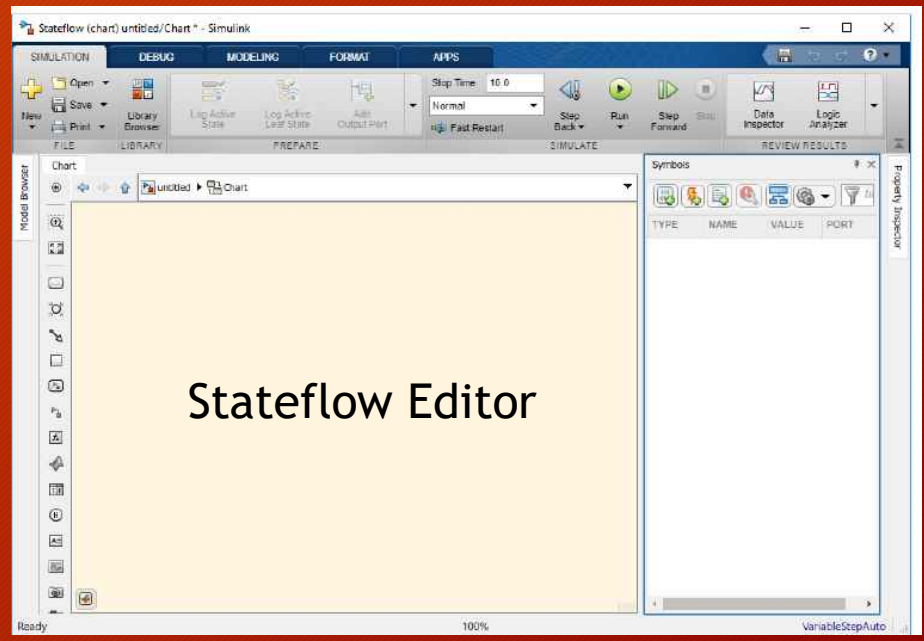
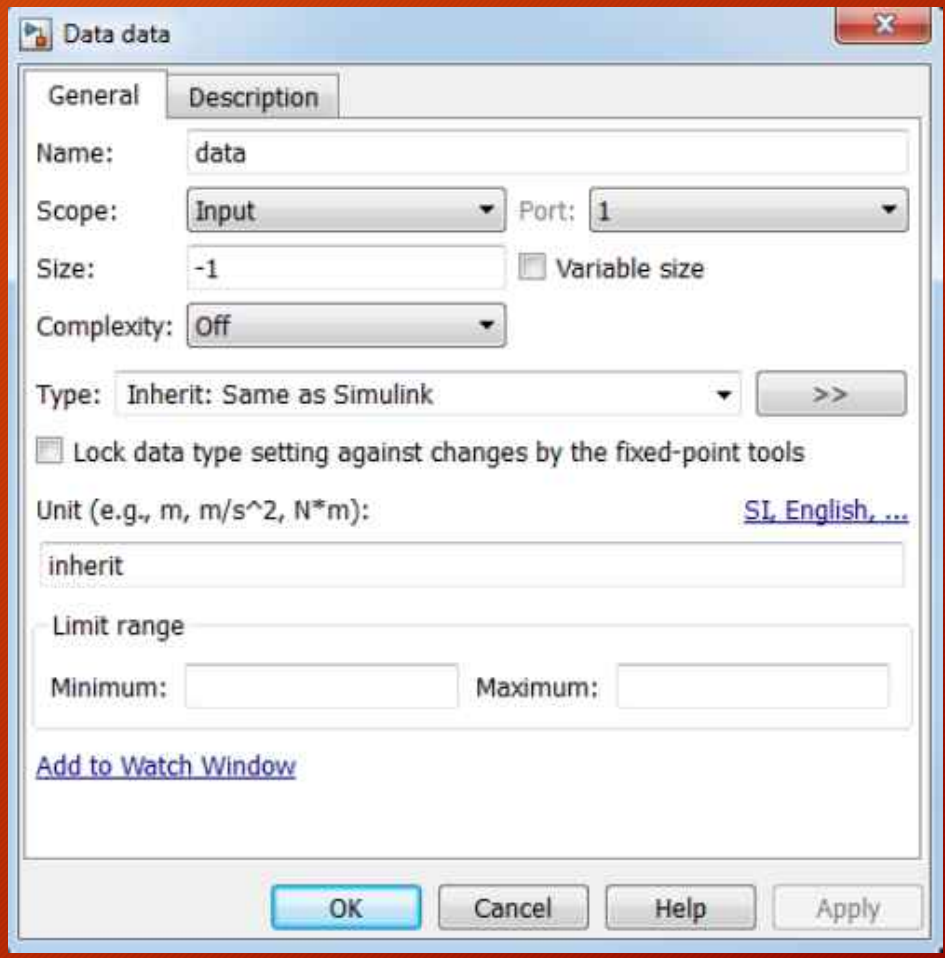


Chart Inputs and Outputs (continued)

2. Add a data element to hold the value of the inputs from, and outputs to the Simulink model:
   - In the Modeling tab, under Design Data, select Data Input
   - The Data properties dialog box opens on your desktop with the General tab selected:
     - The default values in the dialog box depend on the scope — in this case, a data input
   - In the Name field, supply the name of the data element
   - Leave the remaining fields at their default values in the General tab because they meet the requirements for most simple designs, as is

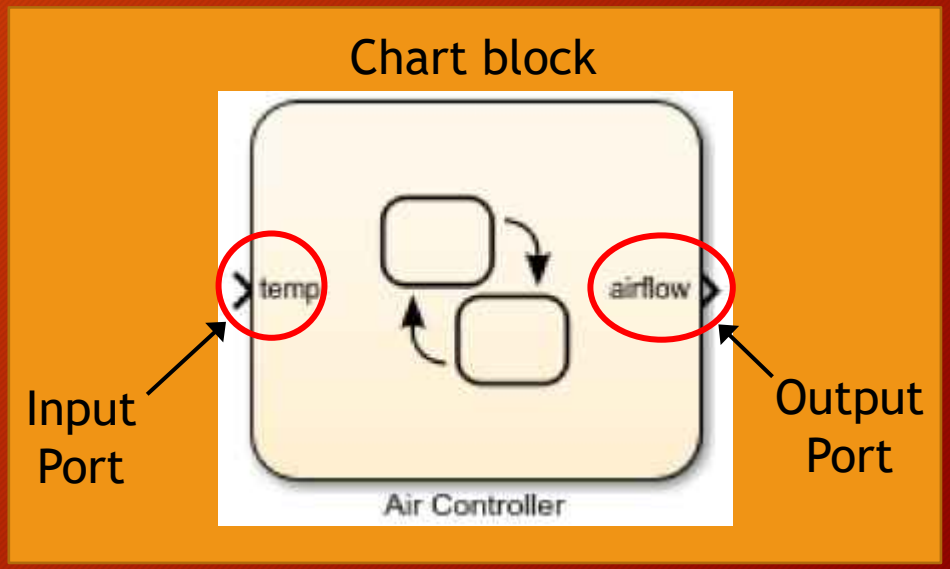


Only an input example is shown here, but the procedure is similar for both inputs and outputs

Inputs, Outputs, and Port Assignments

Field	Default Value	What It Means
Scope	Input	Input from Simulink model. The data element gets its value from the Simulink signal on the same input port.
Size	-1	The data element inherits its size from the Simulink signal on the same port.
Complexity	Off	The data element does not contain any complex values.
Type	Inherit: Same as Simulink	The data element inherits its data type from the Simulink signal on the same output port.

- Ports are assigned to inputs and outputs in the order they are created
- When the first *input* is created, it is assigned to *input port* 1
- You can rearrange port order assignments by simply changing the port numbers
- As each data item is reassigned to a different port, the rest of the data items are rearranged automatically



Section 3

Stateflow Introductory Concepts

Stateflow Charts

- *Finite State Machine*
  - Is a representation of an event-driven (reactive) system
  - In an event-driven system, the system responds to an event by making a transition from one state (mode) to another
  - This transition occurs if the condition defining the change is met
- A *Stateflow chart* is a graphical representation of a Finite State Machine
  - *States* and *Transitions* form the basic elements of the system
- Examples
  - Physical Plant
    - Control the plant in response to events
    - Events could include: temperature and pressure sensors, clocks, and user-driven events
  - Automatic Transmission of a Car
    - The transmission has these operating states: park, reverse, neutral, drive, and low
    - As the driver shifts from one position to another, the system makes a transition from one state to another, for example, from park to reverse

Chart in Simulink Model

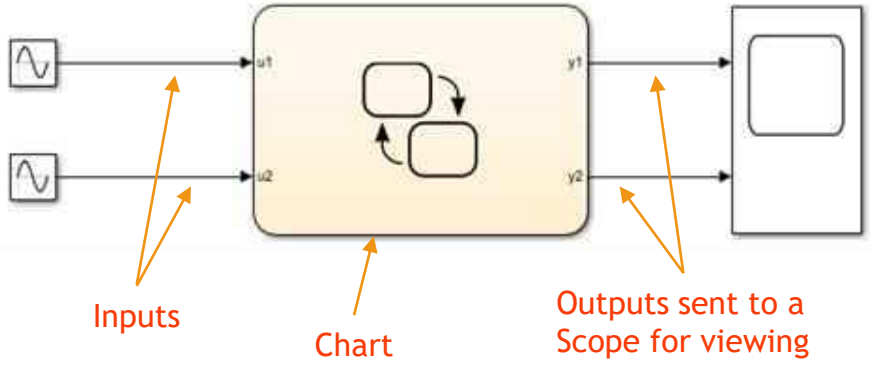
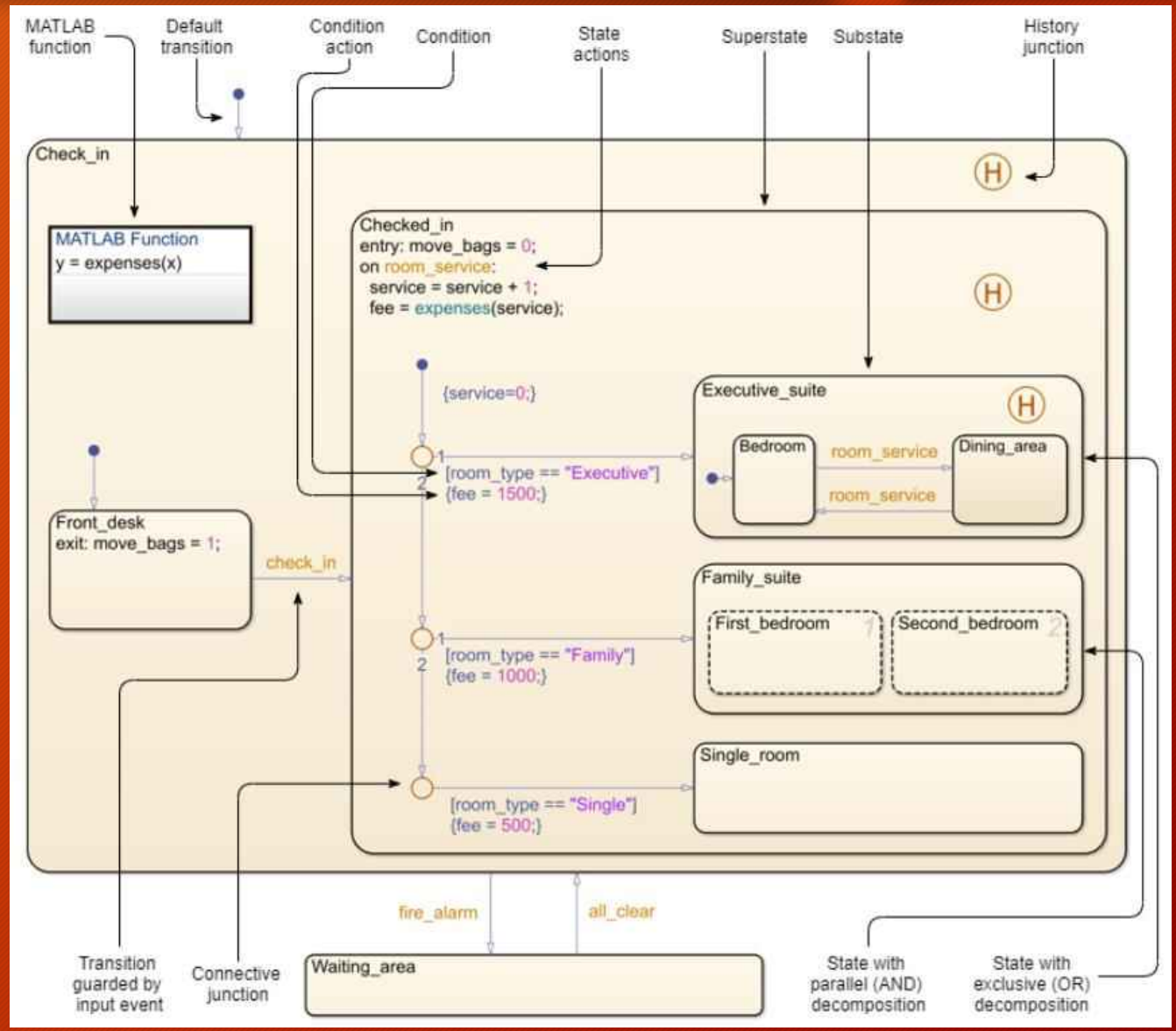


Chart opens into a Stateflow model with detailed states and transitions behavior modeling

Stateflow Objects

- Stateflow objects are the building blocks of Stateflow charts
- These objects can be categorized as either graphical or nongraphical
- Graphical objects consist of objects that appear graphically in a chart
- Nongraphical objects appear textually in a chart and often refer to data, events, and messages
- The figure here shows a variety of both graphical and nongraphical objects used in Stateflow

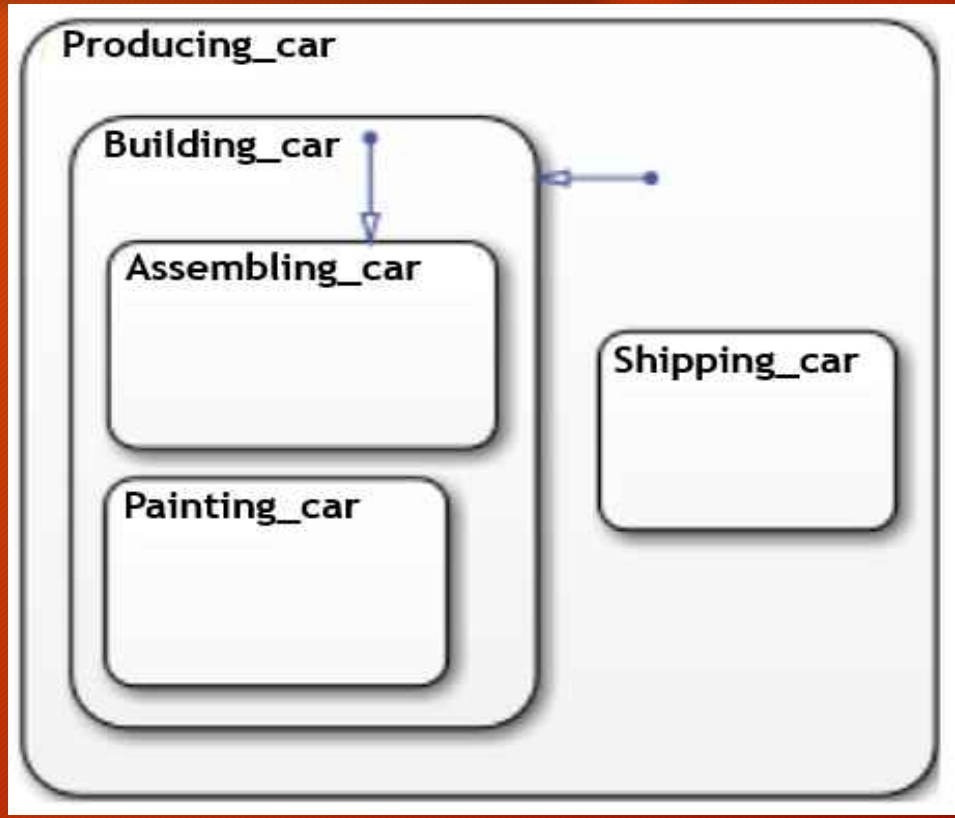


Waking Up a Chart

- A Stateflow chart wakes up:
  - At each time step according to the Simulink solver
  - When the Stateflow chart receives an event
- A chart “wakes up” when it is first activated
- When a chart wakes up for the first time, the chart is initialized and becomes active
- Every time a Stateflow chart wakes up, the chart follows a workflow and executes actions
- Once the chart is active but with no more actions to take, the chart goes to sleep until it is triggered to wake up by a new time step or another event

State Hierarchy

- Establishing a hierarchy among states allows you to
  - Manage multilevel state complexity
  - Represent multiple levels of subcomponents in a system
- In the example, three levels of hierarchy are shown
  - Drawing one state within the boundaries of another state indicates that the inner state is a substate (or child) of the outer state (or superstate)
  - The outer state is the parent of the inner state
- Here, the chart is the parent of the state Producing_car
  - The state Producing_car is the parent state of the Building_car and Shipping_car states
  - The state Building_car is also the parent of the Assembling_car and Painting_car states
  - You can also say that the states Assembling_car and Painting_car are children of the Building_car state
- To represent the Stateflow hierarchy textually, use a slash character (/) to represent the chart and use a period (.) to separate each level in the hierarchy of states

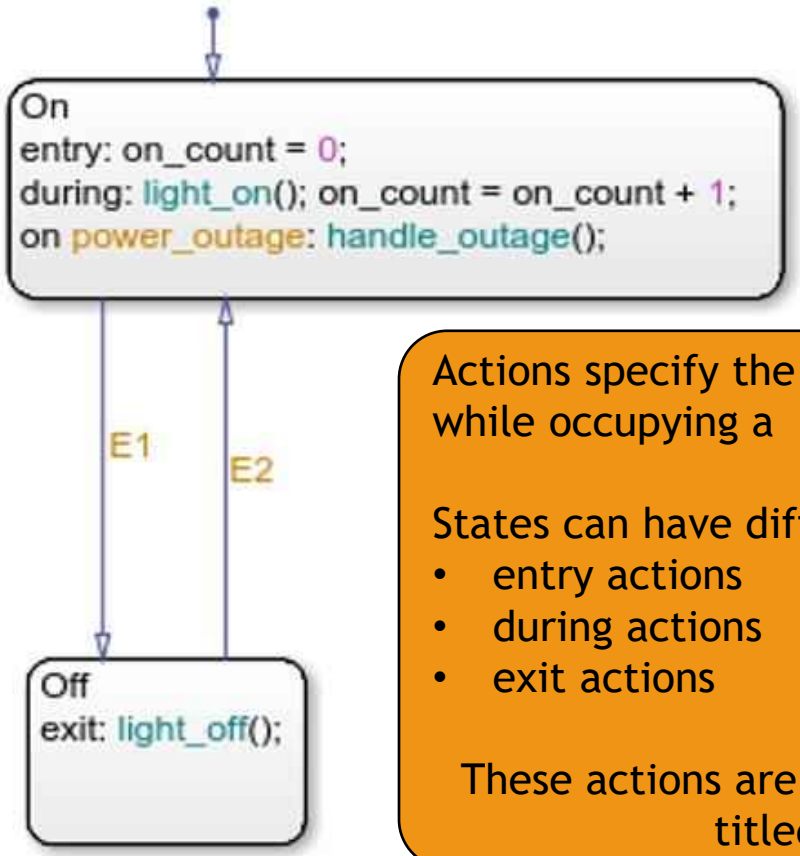


Textual representation of the hierarchy:
/Producing_car
/Producing_car.Building_car
/Producing_car.Shipping_car
/Producing_car.Building_car. Assembling_car
/Producing_car.Building_car.Painting_car

State Labels

- The label for a state appears on the top left corner of the state roundangle (a rounded rectangle) with the following general format:
  - name/
  - entry:entry actions
  - during:during actions
  - exit:exit actions
  - on event_name:on event_name actions
  - on message_name:on message_name actions
  - bind:events

Examples of State Labels



Actions specify the functionality that is performed while occupying a state or during a transition

States can have different action types

- entry actions
- during actions
- exit actions
- bind actions
- on actions

These actions are further defined in the section titled “State Actions”

Events

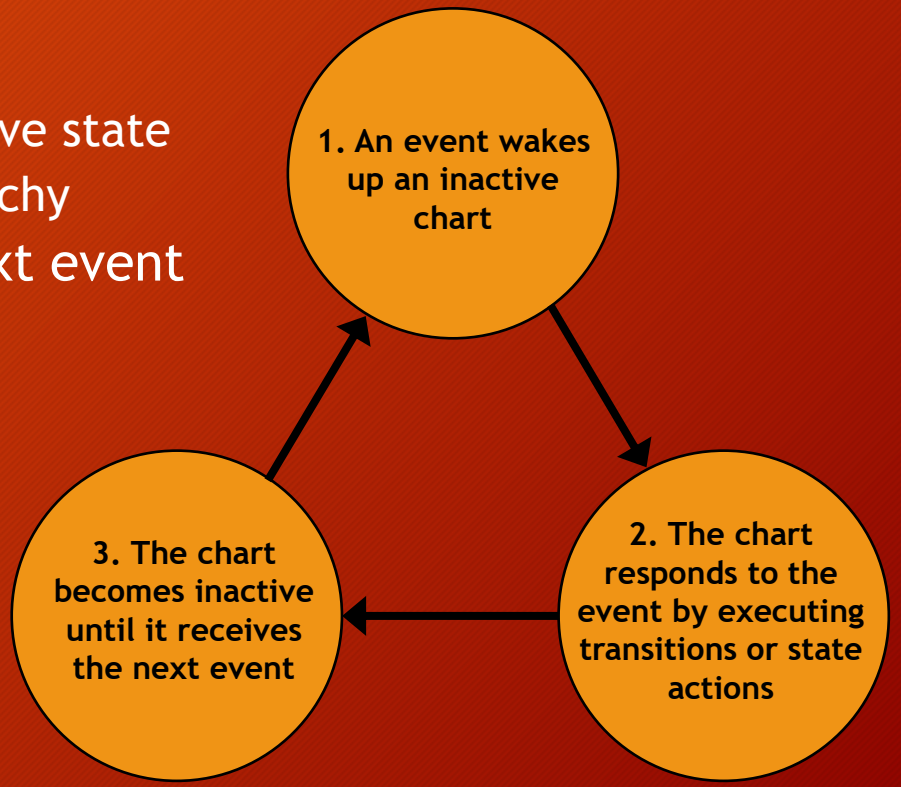
- An event is a nongraphical Stateflow object that can wake up a Stateflow chart
- An event is a nongraphical Stateflow object that can trigger actions in one of these objects:
  - A parallel state in a Stateflow chart
  - Another Stateflow chart
  - A Simulink triggered or function-call subsystem
- Implicit Events
  - An implicit event is a built-in event that is broadcast during chart execution
  - These events are implicit because you do not define or trigger them explicitly
- Explicit Events
  - An explicit event is an event that you define explicitly
  - Explicit events can have one of the three types shown in the table

A state can broadcast an event to any other state that is set up to receive the event

Type of Event	Description
Input Event	An event that is broadcast to a Stateflow chart from outside the chart
Local Event	An event that can occur anywhere in a Stateflow chart but is visible only in the parent object and its descendants Local events are supported only in Stateflow charts in Simulink models
Output Event	An event that occurs in a Stateflow chart but is broadcast to a Simulink block Output events are supported only in Stateflow charts in Simulink models

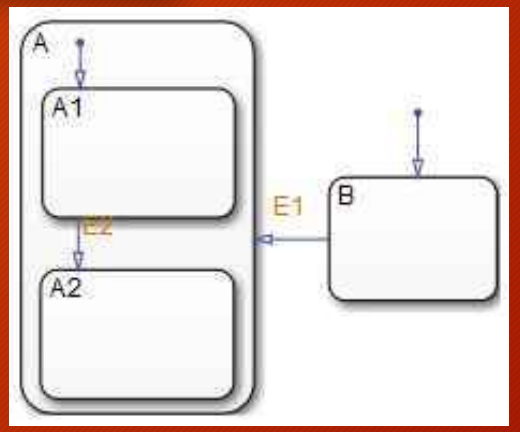
How Stateflow Charts Respond to Events

- Stateflow charts respond to events in a cyclical manner
  - An event wakes up an inactive chart
    1. The chart responds to the event by executing transitions and state actions from the top down through the chart hierarchy
    2. Starting at the chart level:
       i. The chart checks for valid transitions between states
       ii. The chart executes during and on actions for the active state
       iii. The chart proceeds to the next level down the hierarchy
    i. The chart becomes inactive until it receives the next event

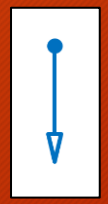


Transactions and Junctions

- *Transition*
  - A line with an arrowhead that links one graphical object to another
  - In most cases, a transition represents the passage of the system from one mode (state) to another
  - A transition typically connects a source and a destination object
  - The source object is where the transition begins and the destination object is where the transition ends
- *Default Transition*
  - A special type of transition that has no source object
- *Junction*
  - Divides a transition into transition segments

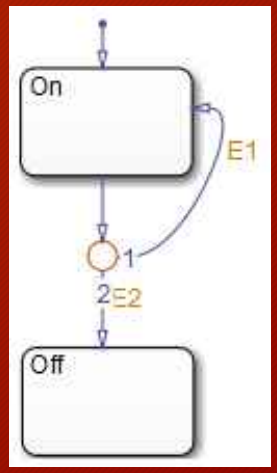


Here, a transition from a source state, B, to a destination state, A



Here, a full transition consists of the segments taken from the origin to the destination state

Each segment is evaluated in the process of determining the validity of a full transition



Additional detail about how transitions function in Stateflow is given in the section titled “State Transitions”

Section 4

Stateflow Charts and States

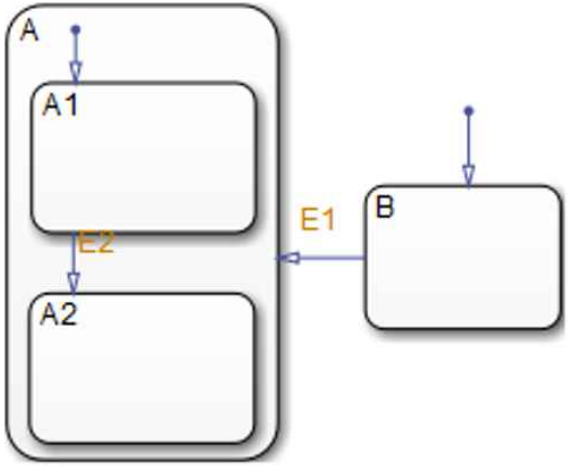
States in Stateflow

- *States*
  - Describe an operating mode of a reactive system
  - In a Stateflow chart, states are used for sequential design to create State Transition Diagrams
  - States can be active or inactive
    - The activity or inactivity of a state can change depending on events and conditions
    - The occurrence of an event drives the execution of the state transition diagram by making states become active or inactive
    - At any point during execution, active and inactive states exist
- States are not Functions
  - States represent a “state of being” for the system – the system is in one state or another at a given time
  - Functions can be executed while in a state
  - Functions can be executed when transitioning from one state to another
  - But, states are not functions

State Hierarchy

- State Hierarchy
  - Hierarchy of states can be developed in a Stateflow chart to manage multilevel state complexity
  - Multiple levels of subcomponents in a system can represent by using state hierarchy
- States can contain all other Stateflow objects
  - Stateflow chart notation supports the representation of graphical object hierarchy in Stateflow charts with containment
  - A state is a *superstate* if it contains other states
  - A state is a *substate* if it is contained by another state
  - A state that is neither a superstate nor a substate of another state is a state whose parent is the Stateflow chart itself
  - States can also contain nongraphical data, event, and message objects
  - The hierarchy of this containment appears in the Model Explorer
  - You define data, event, and message containment by specifying the parent object

State Hierarchy



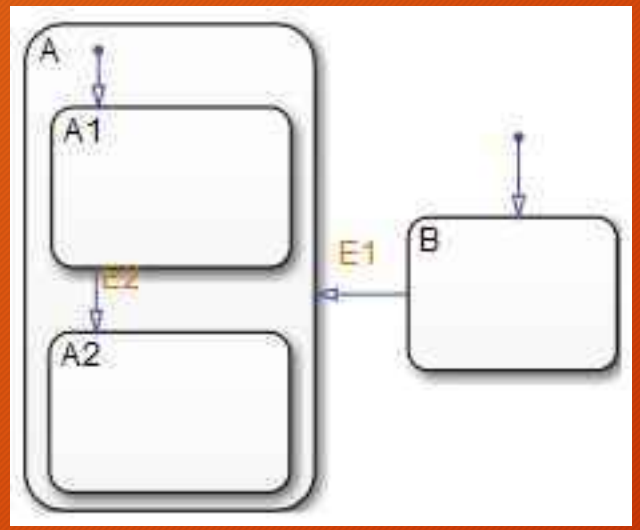
- State A is a superstate
- States A1 and A2 are substates
- The parent of States A and B is the Stateflow chart itself

State Decomposition

- Every state (or chart) has a decomposition that dictates what type of substates the state (or chart) can contain
  - All substates of a superstate must be of the same type as the superstate decomposition
- State decomposition can be one of two types
  - Exclusive (OR) State Decomposition
  - Parallel (AND) State Decomposition

Exclusive (OR) State Decomposition

- Substates with solid borders indicate exclusive (OR) state decomposition
  - Use this decomposition to describe operating modes that are mutually exclusive
  - When a state has exclusive (OR) decomposition, only one substate can be active at a time

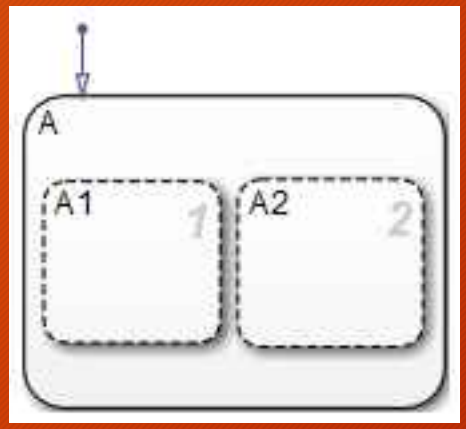


Here, either state A or state B can be active

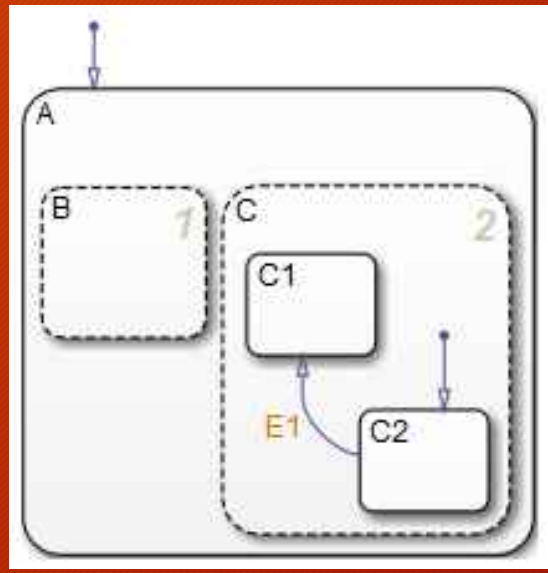
If state A is active, either state A1 or state A2 can be active at a given time

Parallel (AND) State Decomposition

- Substates with dashed borders indicate parallel (AND) decomposition
  - Use this decomposition to describe concurrent operating modes
  - When a state has parallel (AND) decomposition, all substates are active at the same time
- The activity within parallel states is essentially independent



Here, when state A is active, A1 and A2 are both active at the same time



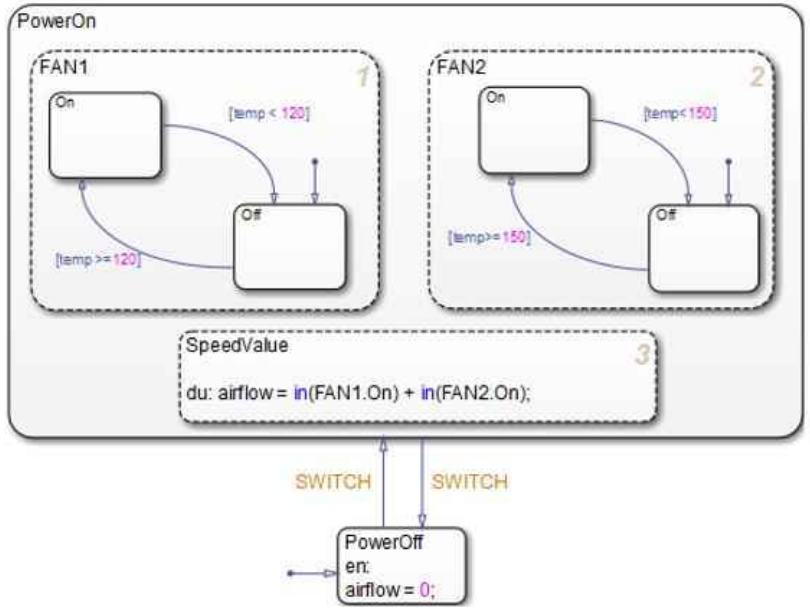
Here, when state A becomes active, both states B and C become active at the same time

When state C becomes active, either state C1 or state C2 can be active, bot not both at the same time (Exclusive OR)

State Names

- A *State Label* starts with the name of the state followed by an optional / character
- A valid *State Name* consists of alphanumeric characters and can include the underscore (_) character
- The name that you enter on the state label must be unique when preceded by ancestor states
- Hierarchy provides some flexibility in naming states
- Each state can have the same name appear in the label, as long as their full names within the hierarchy are unique
- The name in the Stateflow hierarchy is the text you enter as the label on the state, preceded by the names of parent states separated by periods

State Names



- Each of these states has a unique name because of its location in the chart
- The full names for the states in FAN1 and FAN2 are:
  - PowerOn.FAN1.On
  - PowerOn.FAN1.Off
  - PowerOn.FAN2.On
  - PowerOn.FAN2.Off

Entering a Chart

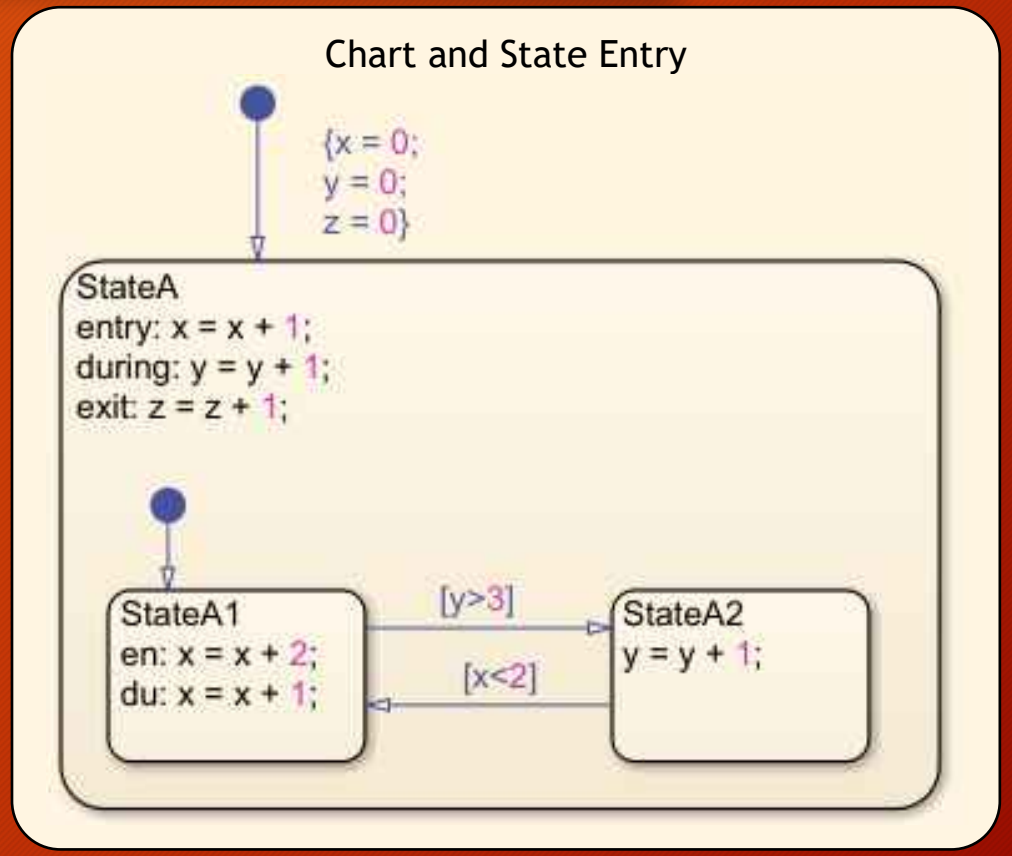
- Chart entry occurs when:
  - A chart is activated for the first time (called *chart initialization*)
- On initialization, the chart is entered and Stateflow executes any default transitions for exclusive (OR) states
- If the states at the top level of your chart are parallel (AND), they become active based on their order number
- For the chart to take default transitions before time t = 0
  - In the Chart Properties dialog box, select the Execute (enter) chart at initialization check box
  - This option causes the Stateflow chart to initialize at the same time as Simulink initialization
  - The default transition paths of the chart then execute during the model initialization phase

Entering a State

- State entry occurs when:
  - A valid transition into a state exists
- Once the chart is active and has gone through initialization, the top-level state becomes active
- A state is marked for entry in one of these ways:
  - An incoming transition crosses state boundaries
  - An incoming transition ends at the state boundary
  - The state is a parallel state child of an active state
- When a state is marked for entry, the entry actions for a state execute

Chart and State Entry Example

- Time Step 0
  - The default transition actions are executed
  - Now x = 0, y = 0, and z = 0
  - StateA is marked for entry
    - The entry actions for StateA are performed
    - Now x = 1, y = 0, and z = 0
  - There is a default transition to the substate, StateA1
  - StateA1 is marked for entry
    - The entry actions for StateA1 are performed
    - Now x = 3, y = 0, and z = 0
- Time Step 1
  - The during actions are performed for StateA
  - Now x = 3, y = 1, and z = 0
  - The during actions are performed for StateA1
  - Now x = 4, y = 1, and z = 0

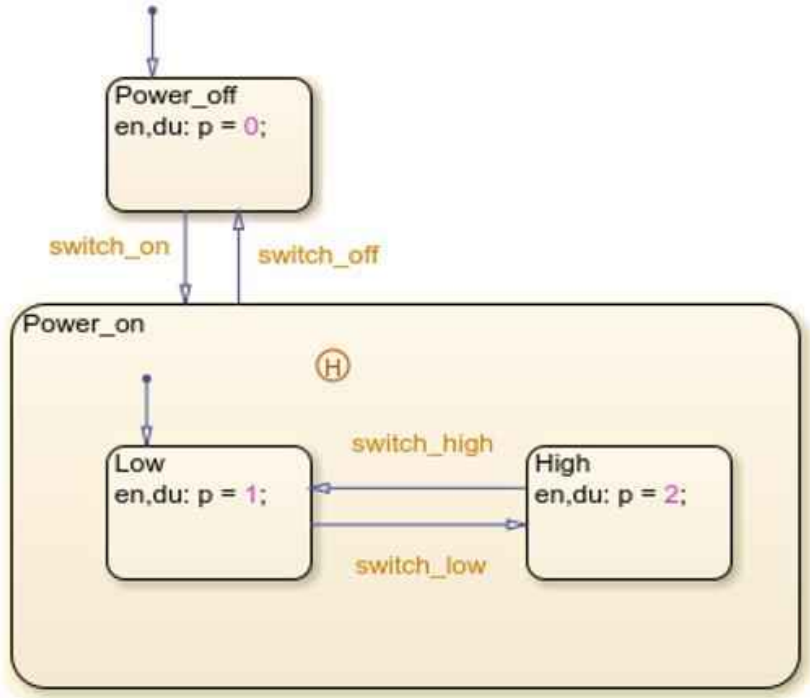


- Time Step 2
  - The during actions are performed for StateA
  - Now x = 4, y = 2, and z = 0
  - The during actions are performed for StateA1
  - Now x = 5, y = 2, and z = 0

Entering a State by Using History Junction

- To have a Stateflow chart remember and return to a substate that was previously active, regardless of a default transition, use a *History Junction* (H)
- Placing a history junction within a state overrides the default transition leading to exclusive (OR) substates
- After placing a history junction within a state, upon entry, your Stateflow chart remembers and enters the previously active substate
- The history junction applies only to the level of the hierarchy in which it appears

History Junction



- A light can be on or off; as indicated by the states *Power_on* and *Power_off*
- The options are controlled by the input events *switch_on* and *switch_off*
- When the light is on, it can be dim or bright
- These options are indicated by the states *Low* and *High* and are controlled by the input events *switch_low* and *switch_high*

- Initially, the chart is asleep
- The state *Power_off* is active
- When the state *Power_on* was last active, *High* was the previously active substate
- The event *switch_on* occurs and the state *Power_on* is marked for entry
- At this time p = 0
- The state *Power_on* becomes active
- Based on the history of *Power_on*, substate High will be entered
- At this time p = 2

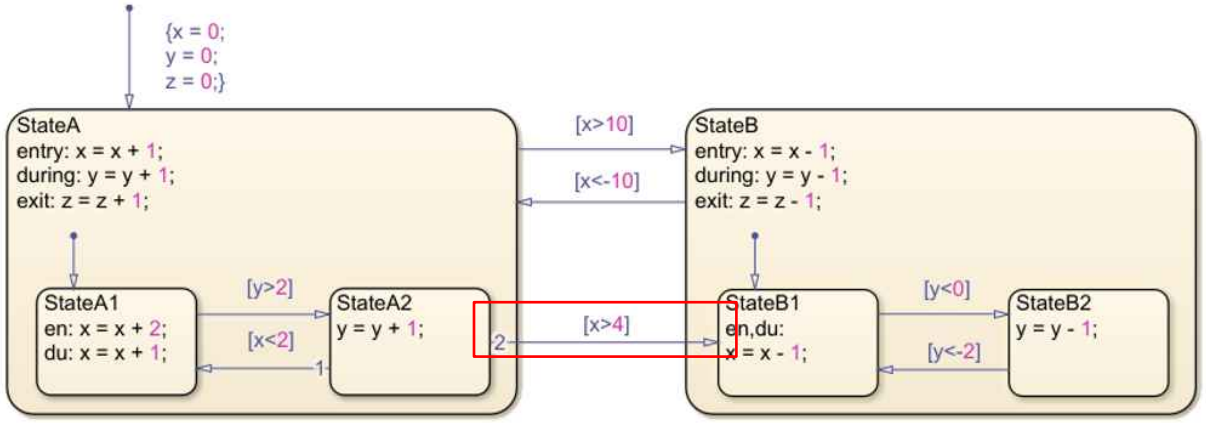
Exiting a State

- When there is a valid transition out of a state, that state is marked for exit
- A state is marked for exit in one of these ways:
  - The outgoing transition originates at the state boundary
  - The outgoing transition crosses the state boundary
  - The destination state is a parallel state child of an activated state
- Exit actions for a state execute when the state is active and a valid transition from the state exists
- A state performs its exit actions before becoming inactive
- Exit actions are preceded by the prefix exit or ex, followed by a required colon (:), and then followed by one or more actions
- To implement multiple actions,, separate them with a carriage return, semicolon (;), or a comma (,)

Exiting/Entering States by Using Supertransitions

- A *Supertransition* is a transition between different levels in a chart
- A supertransition can occur
  - Between a state in a top-level chart and a state in one of its subcharts
  - Between states residing in different subcharts at the same or different levels in a chart
- Supertransitions can span any number of levels in a chart
- When a state is exited through a supertransition, after the exit actions for the source of the transition are executed (StateA2), its superstates are marked inactive and exit actions of the superstates are executed (StateA)
- When a state is entered through a supertransition, before the entry actions for the final destination are executed (StateB1), its superstates must be marked active and their entry actions must be executed (StateB)

Supertransitions



- The system is currently in StateA2
  - At this point, x = 5, y = 5, and z = 1
- StateA2 is marked for exit
  - The exit actions for StateA2 are performed (none)
- StateA is the parent of StateA2
  - The exit actions for StateA are performed
  - Now, x = 5, y = 5, and z = **2**
- StateB1 is marked for entry from StateA2
- StateB is the parent of StateB1
- The entry actions for StateB are performed
  - Now, x = **4**, y = 5, and z = 2
- The entry actions for StateB1 are performed
  - Now, x = **3**, y = 5, and z = 2

Section 5

State Actions

State Labels

- The label for a state appears on the top left corner of the state rectangle
- Actions of different types are enterd on separate lines after the name of the state
  - Actions are optional
  - You can enter these actions in any order

Format of a State Label

```
name/
entry:entry actions
during:during actions
exit:exit actions
on event_name:on event_name actions
on message_name:on message_name actions
bind:events
```

Actions specify the functionality that is performed while occupying a state or during a transition

entry - executed when a state becomes active

exit - executed when a state is active and a transition out of the state occurs

during - executed when a state is active, an event occurs, and no valid transition is available

bind - bind the data and events to a state

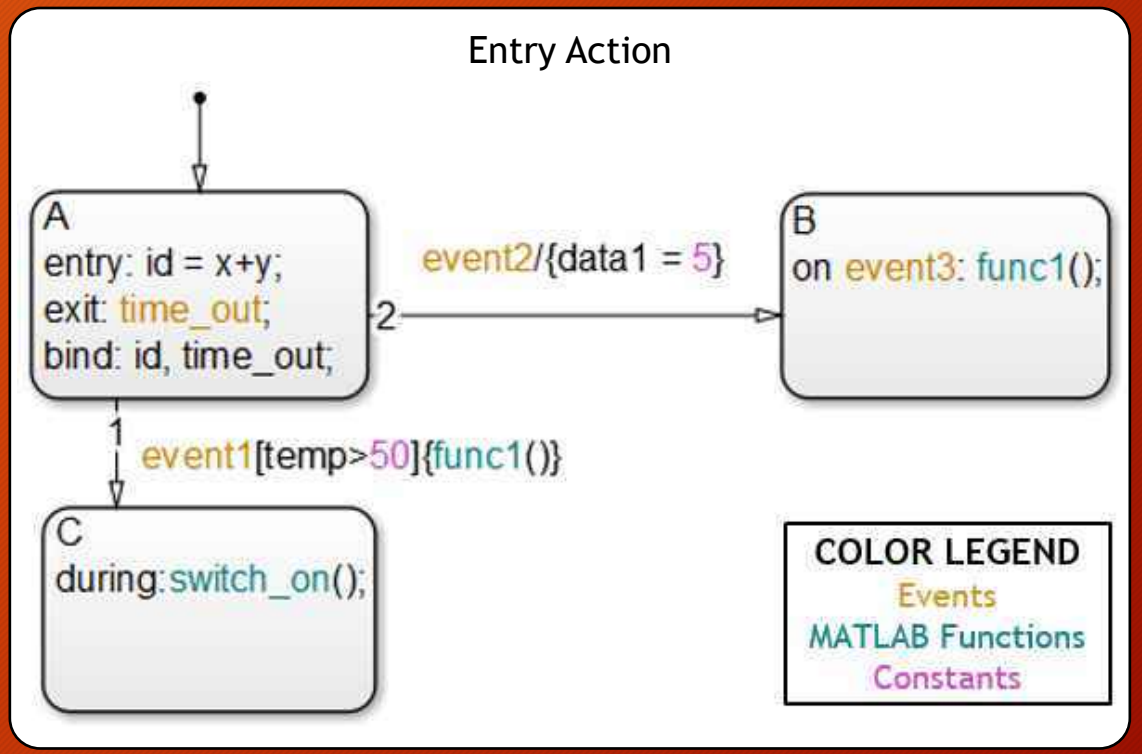
on - executed when the state is active and it receives an event or message

State Actions

- Action statements are optional and are placed after the state label, with a keyword label that identifies the type of action
  - You can specify none, some, or all of them
  - The colon after each keyword is required
  - The slash following the state name is optional and is normally followed by a carriage return if action statements follow
    - If you enter the name and slash followed directly by actions (no carriage return), the actions are interpreted as entry actions
    - This shorthand is useful if you are specifying only entry actions
    - If you do not specify the action type explicitly for a statement, the chart treats that statement as a combined entry-during action
- WIth each type of action, you can enter more than one action by separating each action with a carriage return, semicolon, or a comma
- You can specify actions for more than one event or message by adding additional *on event_name* or *on message_name* lines

Entry Action

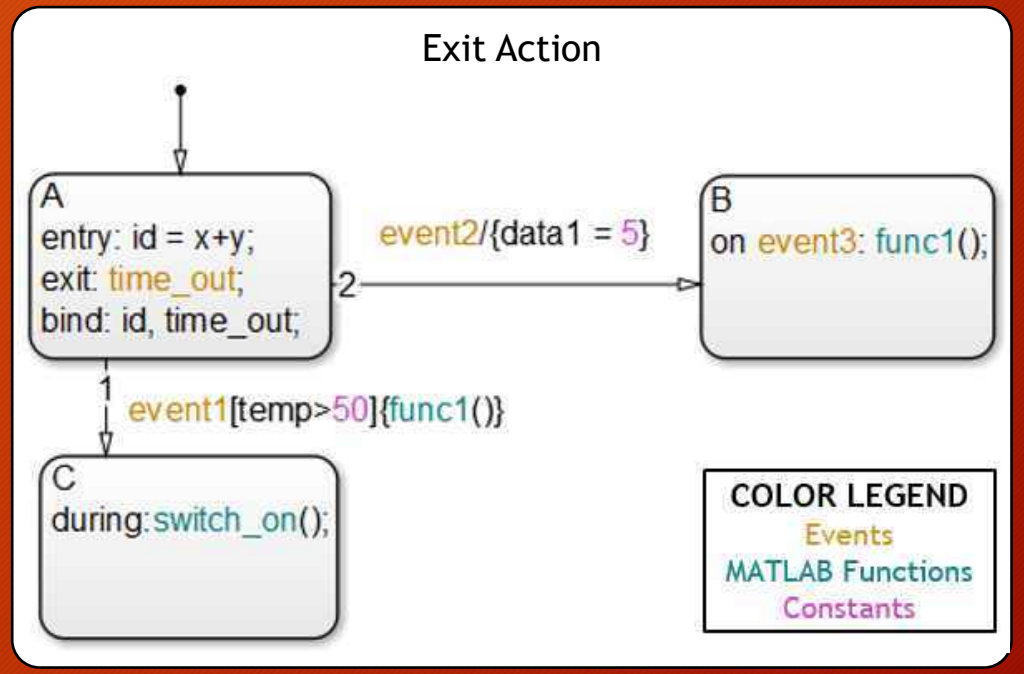
- Entry actions are executed when a state becomes active
- Entry actions consist of the prefix *entry* (or the abbreviation *en*) followed by a colon (:) and one or more actions
  - To separate multiple entry actions, use semicolons or commas
  - Additional actions can also be entered on separate lines



The entry action *id = x+y* executes when the chart takes the default transition and state A becomes active

Exit Action

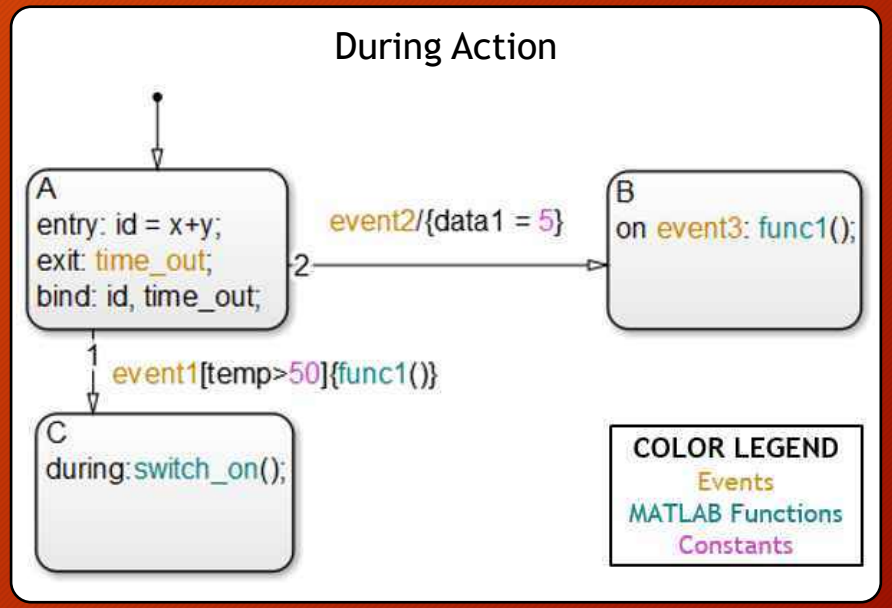
- Exit actions are executed when a state is active and a transition out of the state occurs
- Exit actions consist of the prefix *exit* (or the abbreviation *ex*) followed by a colon (:) and one or more actions
  - To separate multiple exit actions, use semicolons or commas
  - Additional actions can also be entered on separate lines



The exit action *time_out* executes when the chart takes one of the transitions from state A to state B or C

During Action

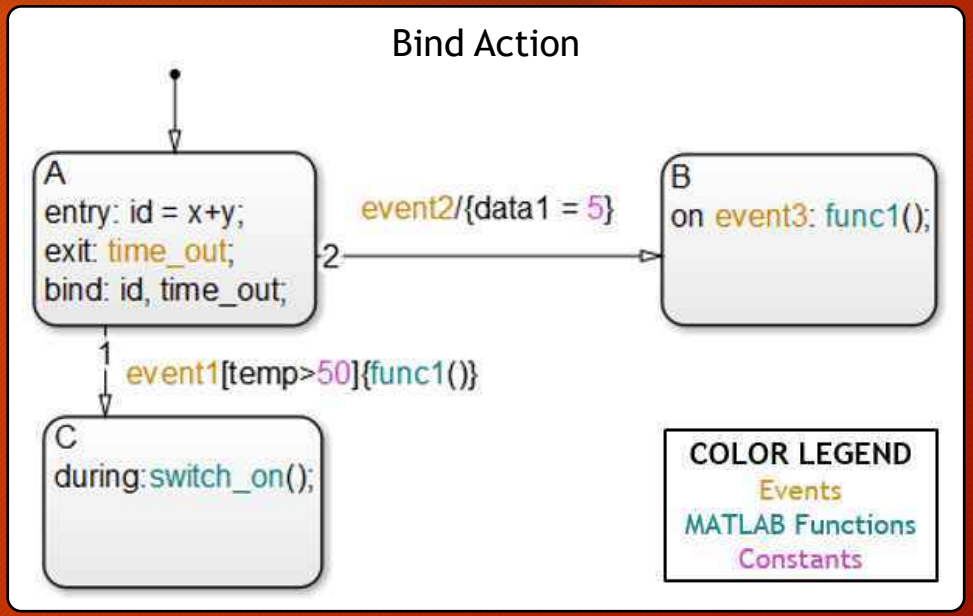
- During actions for a state execute when:
  - The state is active, a new time step occurs, and no valid transition to another state is available
  - The state is active, an event occurs, and no valid transition to another state is available
- During actions consist of the prefix *during* (or the abbreviation *du*) followed by a colon (:) and one or more actions
  - To separate multiple during actions, use semicolons or commas
  - Additional actions can also be entered on separate lines
- State action types that are not specified explicitly for a statement are treated as an *entry,during* action
- A state performs its during actions (if specified) when the chart wakes up
- If a Stateflow chart does not contain states, each time the chart is executed, Stateflow always evaluates the default transition path



The during action *switch_on()* executes whenever the state C is active because there are no valid transitions to another state

Bind Action

- You can bind the data and events to a state by using a bind action
- A bind action consists of the prefix *bind* followed by a colon (:) and one or more events or data
  - To separate multiple events and data, use semicolons or commas
  - Additional actions can also be entered on separate lines
- Only a state and its children can change data or broadcast events bound to that state
- Other states can read the bound data or listen for the bound event, but they cannot change the bound data or send the bound events
- Bind actions apply to a chart whether the binding state is active or not
- If a state includes actions that change data or broadcast events that bind to another state, a compile-time parsing error occurs

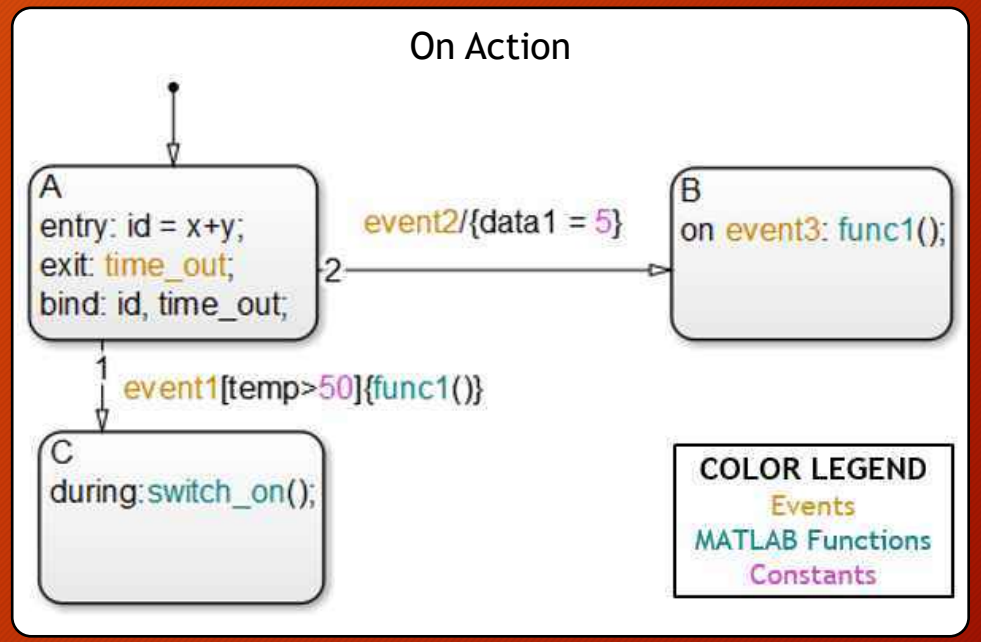


Here, the bind action *bind: id, time_out* for state A binds the data *id* and the event *time_out* to state A

This binding prevents any other state (or its children) in the chart from changing *id* or broadcasting event *time_out*

On Action

- On actions are executed when the state is active and it receives an event or message
- On actions consist of the prefix *on* followed by a unique event event_name or message message_name, a colon (:), and one or more actions
  - To separate multiple on actions, use semicolons or commas
  - Additional actions can also be entered on separate lines
- Actions can be specified for more than one event or message
  - To have different events trigger different actions, enter multiple on action statements in the state action label:
    - on ev1: action1();
    - on ev2: action2();
  - If multiple events occur at the same time, the corresponding on actions execute in the order that they appear in the state action label
  - If events *ev1* and *ev2* occur at the same time, then *action1()* executes first and *action2()* executes second



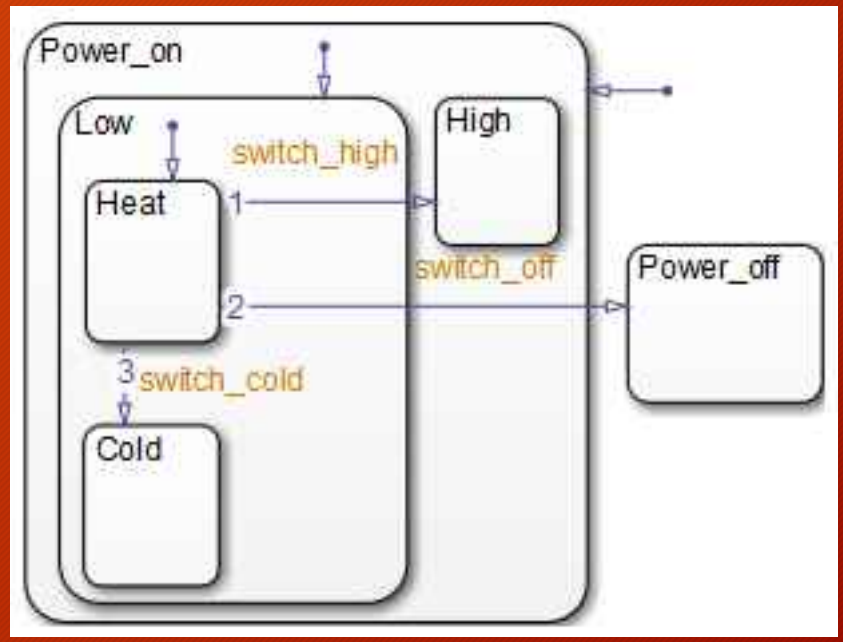
The on action *on event3: funct1()* for state B executes *funct1()* when the state is active and it receives the event *event3*

Section 6

State Transitions

Transition Hierarchy

- Transitions cannot contain other objects the way that states can
  - However, transitions are contained by states
- The hierarchy for a transition is described in terms of its parent, source, and destination states
  - The parent is the lowest level that contains the source and destination of the transition
- The following table resolves the parentage of each transition in the accompanying example
  - The slash character (/) represents the chart
  - Each level in the hierarchy of states is separated by the period character (.)



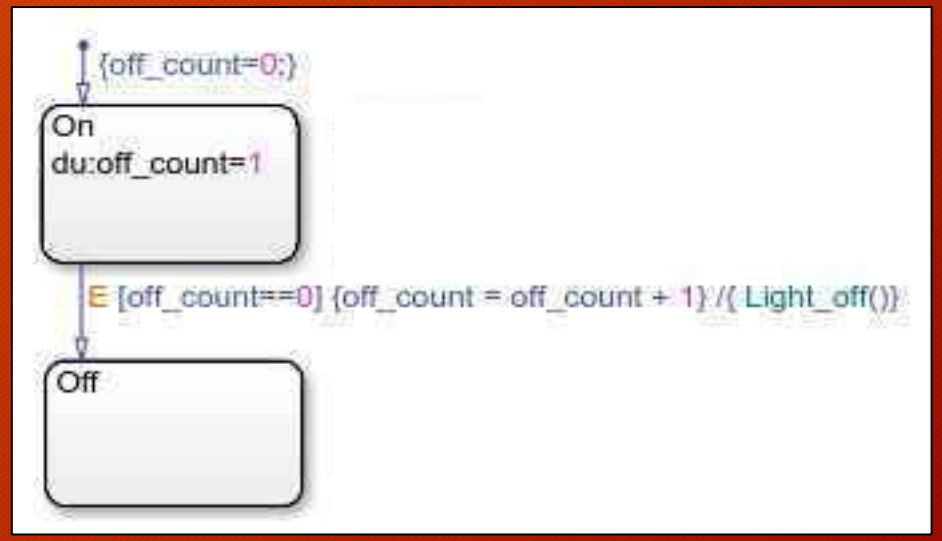
Transition Label	Transition Source	Transition Destination	Transition Parent
switch_off	/Power_on.Low.Heat	/Power_off	/
switch_high	/Power_on.Low.Heat	/Power_on.High	/Power_on
switch_cold	/Power_on.Low.Heat	/Power_on.Low.Cold	/Power_on.Low

Transition Labels

- A transition label can consist of an event or message, a condition, a condition action, and a transition action
  - Each part of the label is optional
  - The ? character is the default transition label
- Transition labels have this overall format:

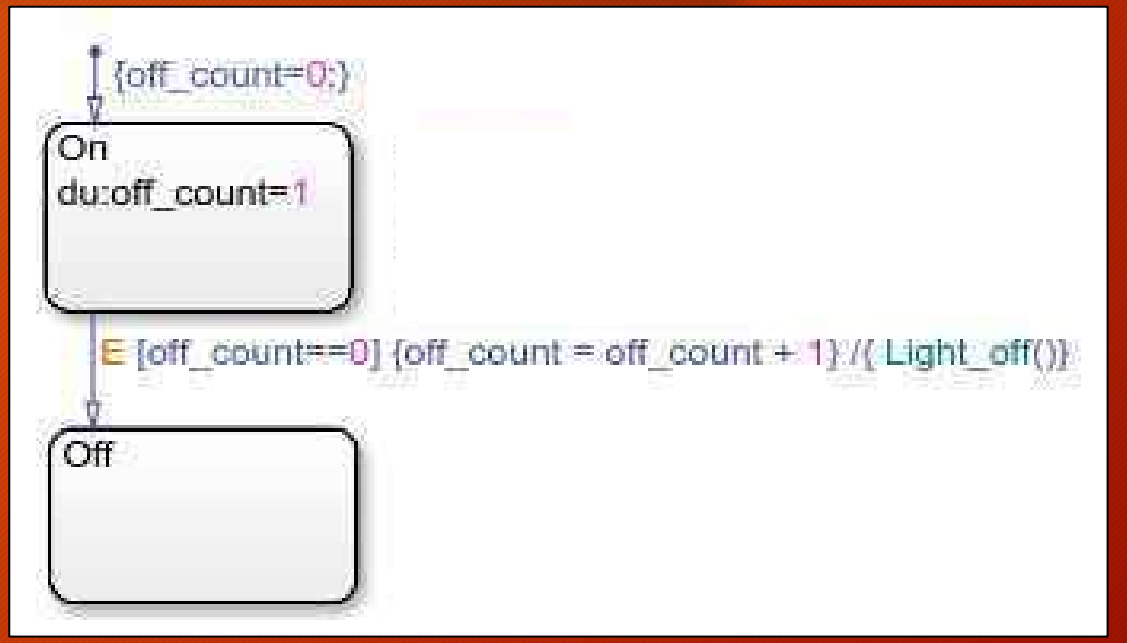
event_or_message[condition]{condition_action}/transition_action

This example illustrates the parts of a transition label which are discussed in detail in the following slides



Elements of Transition Labels

- Event or Message Trigger
  - Specifies an event or message that causes the transition to occur when the condition is true
  - To specify multiple events, use the OR logical operator (|)
  - Specifying an event or message is optional
  - The absence of an event or message indicates that the transition takes place on the occurrence of any event



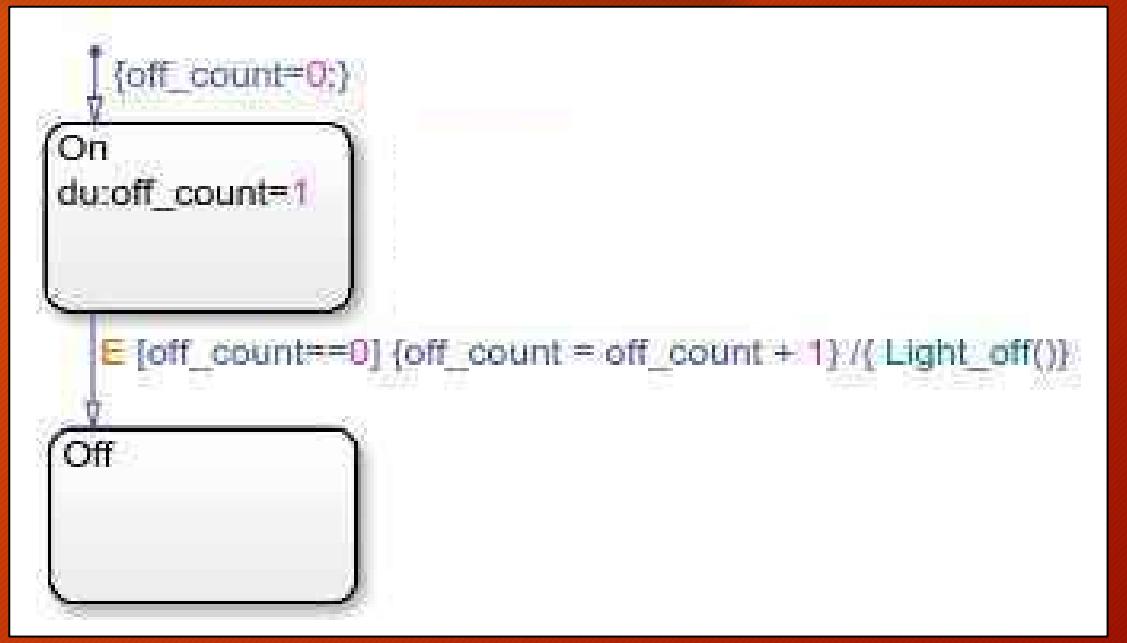
Usage of Transition Label Components in this Example

Event or Message Trigger

In the example, the broadcast of event E triggers the transition from On to Off if the condition [off_count==0] is true

Elements of Transition Labels (continued)

- Condition
  - Specifies a Boolean expression that, when true, validates a transition for the specified event or message trigger
  - Enclose the condition in square brackets ([ ])
  - If no condition is specified, an implied condition evaluates to true



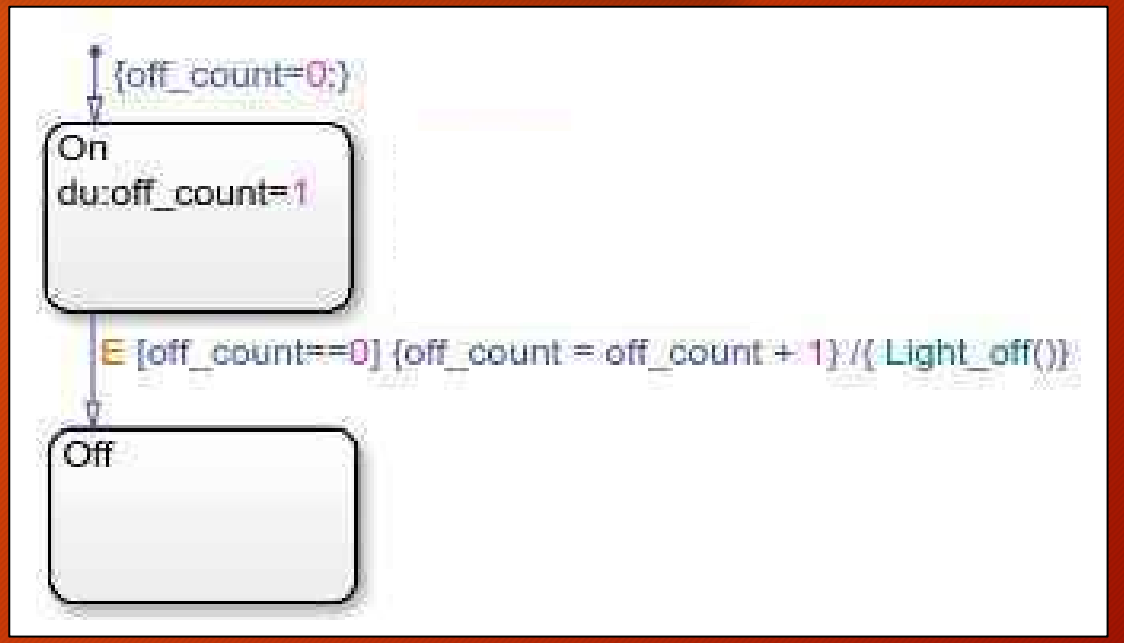
Usage of Transition Label Components in this Example

Condition

In the example, when the event E occurs, the condition [off_count==0] must evaluate as true for the transition from On to Off to be valid

Elements of Transition Labels (continued)

- Condition Action
  - Executes after the condition for the transition is evaluated as true, but before the transition to the destination is determined to be valid
  - Enclose the condition action in curly braces ({ }) following the condition



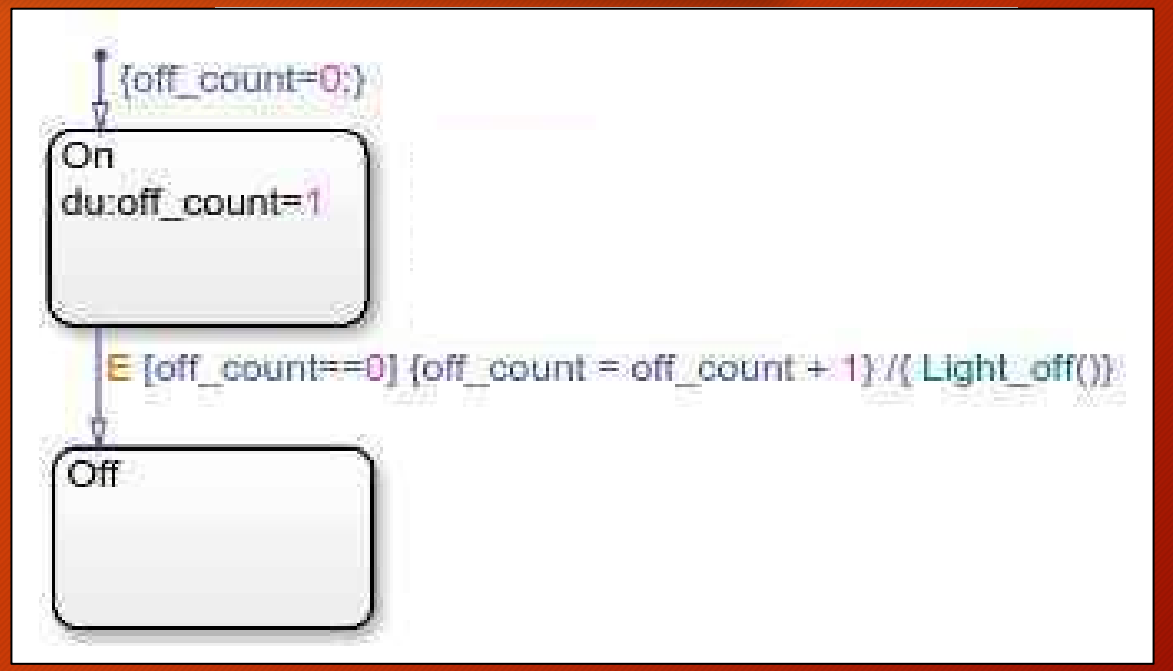
Usage of Transition Label Components in this Example

Condition Action

In the example, if the event E occurs and the condition [off_count==0] is true, then the condition action {off_count = off_count + 1} is immediately executed

Elements of Transition Labels (continued)

- Transition Action (unique to Stateflow)
  - Executes after the transition to the destination is determined to be valid
  - If the transition consists of multiple segments, then the transition action is executed when the entire transition path to the final destination is determined to be valid
  - Transition actions occur after the exit actions of the source state and before the entry actions of the destination state
  - Precede the transition action with a slash character (/)



Usage of Transition Label Components in this Example

Condition

In the example, when the event E occurs, the condition [off_count==0] **Transition Action**

In the example, if the event E occurs and the condition [off_count==0] is true, then the transition action {Light_off()} is executed when the transition from On to Off is determined to be valid

The transition action occurs after On becomes inactive, but before Off becomes active

Valid Transitions

- A transition is usually valid when the source state of the transition is active and the transition label is valid
  - Default transitions are different because there is no source state
  - Validity of a default transition to a substate is evaluated when there is a transition to its superstate, assuming the superstate is active
- This labeling criterion applies to both default transitions and general case transition

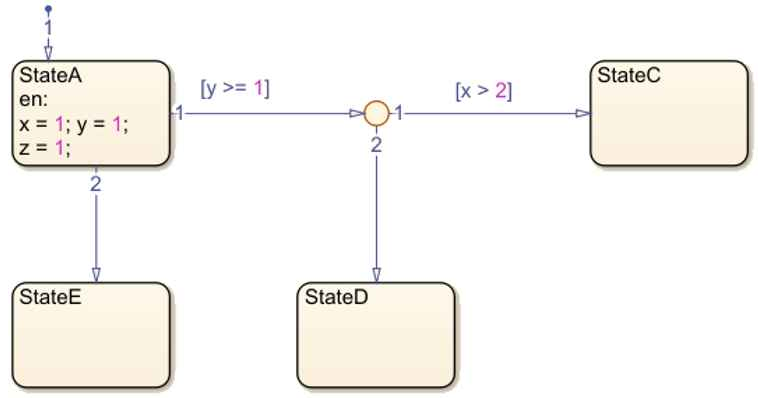
This table lists possible combinations of valid transition labels

Transition Label	Is Valid If...
Event only	That event occurs
Event and condition	That event occurs and the condition is true
Message only	That message occurs
Message and condition	That message occurs and the condition is true
Condition only	Any event occurs and the condition is true
Action only	Any event occurs
Not specified	Any event occurs

Transition Evaluation

- Stateflow uses valid transitions in charts to move from one exclusive (OR) state to another exclusive (OR) state
- A valid transition is a transition whose condition labels are met (evaluate to True) and whose path ends at a state
- If a transition is valid, Stateflow exits from the source state and enters the destination state
- When multiple transitions originate from a single source, such as a state or junction, Stateflow uses evaluation order to determine when to test each transition
- Depending on which action language your chart uses, you can create the order of your transitions explicitly or implicitly
- Whether explicitly or implicitly ordered, transitions show a number near the source of the transition that designates the transition order

Supertransitions

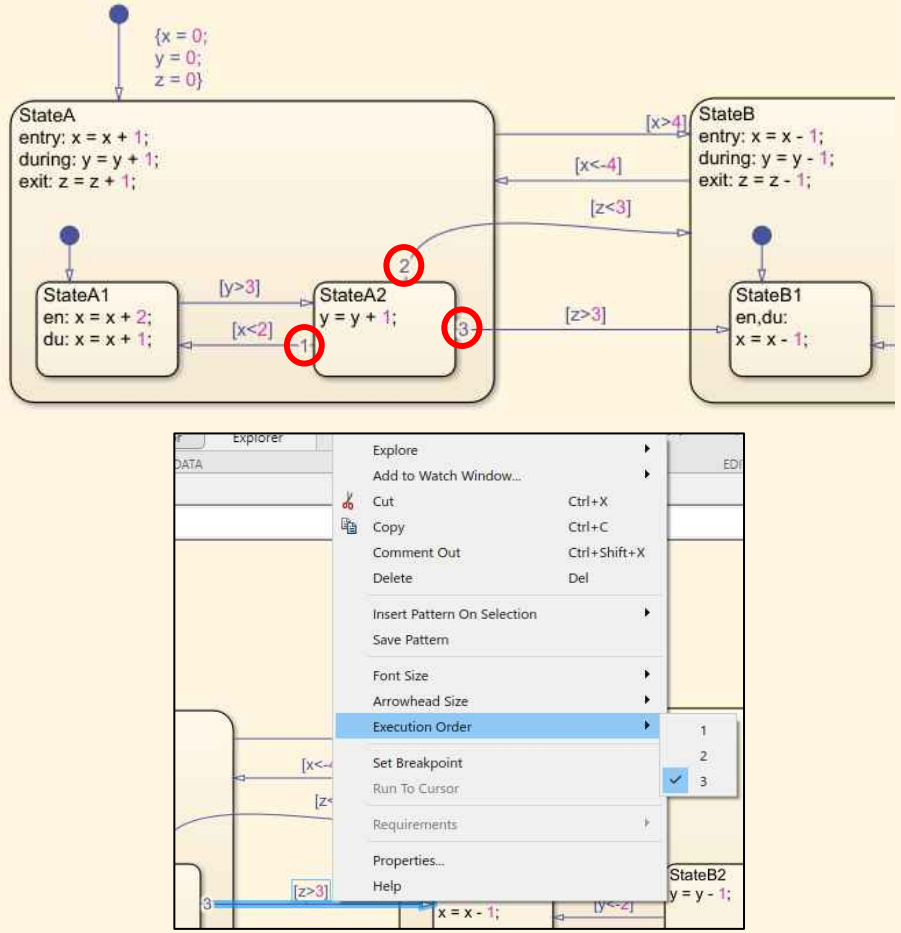


- Two transitions originate from StateA
  - They are marked as 1 and 2
- Two transitions originate from the junction coming from StateA
  - They are marked as 1 and 2

Explicit Transition Ordering

- In a new Stateflow chart, all outgoing transitions from a source are automatically numbered in the order in which you create them
- The order starts with 1 and continues to the next available number for the source
- The execution order of a transition can be changed by the user explicitly
- The other outgoing transitions for the source are automatically renumbered, thus preserving their relative order

Change Transition Order



- Select the transition to be reordered
- Right Click -> Execution Order

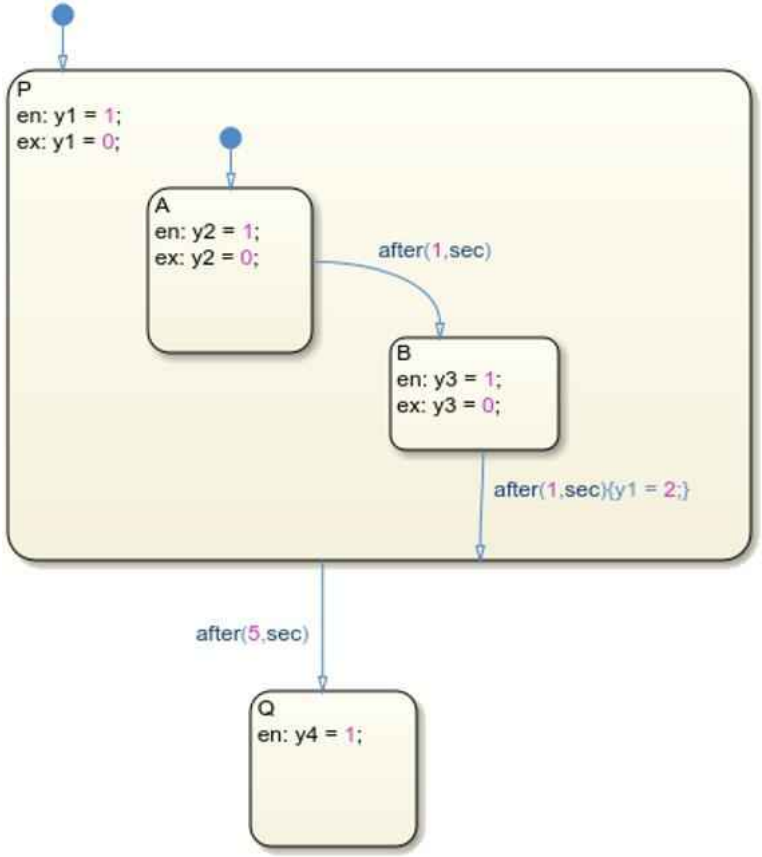
Implicit Transition Ordering

- A group of outgoing transitions from a single source are evaluated based on:
  - Hierarchy
    - A group of outgoing transitions in an order is evaluated based on the hierarchical level of the parent of each transition
  - Label
    - A group of outgoing transitions with equal hierarchical priority based on the labels is evaluated in the following order of precedence:
      - Labels with events and conditions
      - Labels with events
      - Labels with conditions
      - No label
  - Angular surface position of transition source
    - A group of outgoing transitions with equal hierarchical and label priority is evaluated based on the angular position of the transition on the surface of the source object
    - The transition with the smallest clock position (0 to 360 degrees, clockwise for upper center) has the highest priority
    - For example, a transition with a 2 o'clock source position has a higher priority than a transition with a 4 o'clock source position
    - A transition with a 12 o'clock source position has the lowest priority

Transition to the Inner Edge of a Parent State

- Transitions that end on the inside edge of a parent state are a shortcut back to the default transition path, and the default path is evaluated during the current timestep
- If there are default transitions, then Stateflow immediately executes those paths
- If not, and there are no children, then that is the end of the timestep
- In both cases, the parent remains active, and exit and entry actions of the parent are not executed

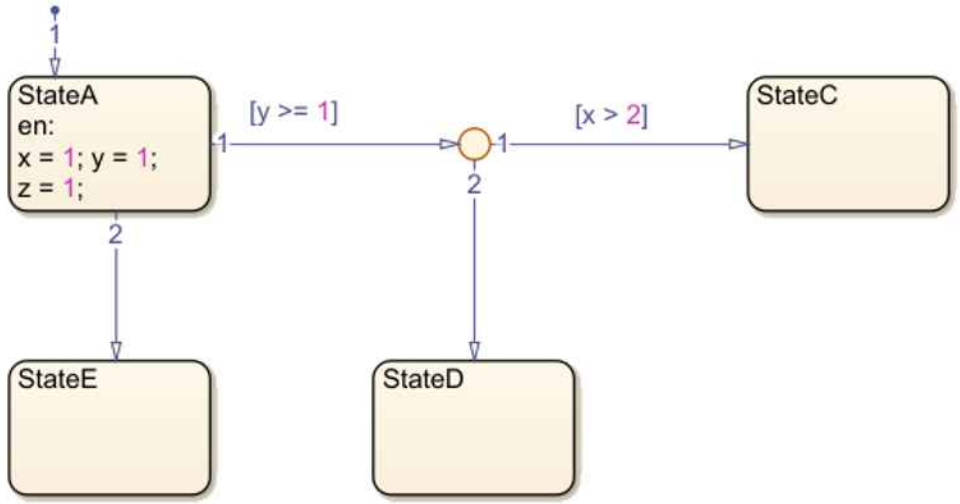
Transition to the Inner Edge of a Parent State



- The transition from state B leads immediately to the default transition to state A

Evaluating an Outgoing Transition with a Junction

Evaluating an Outgoing Transition with a Junction



Transition Evaluation Order

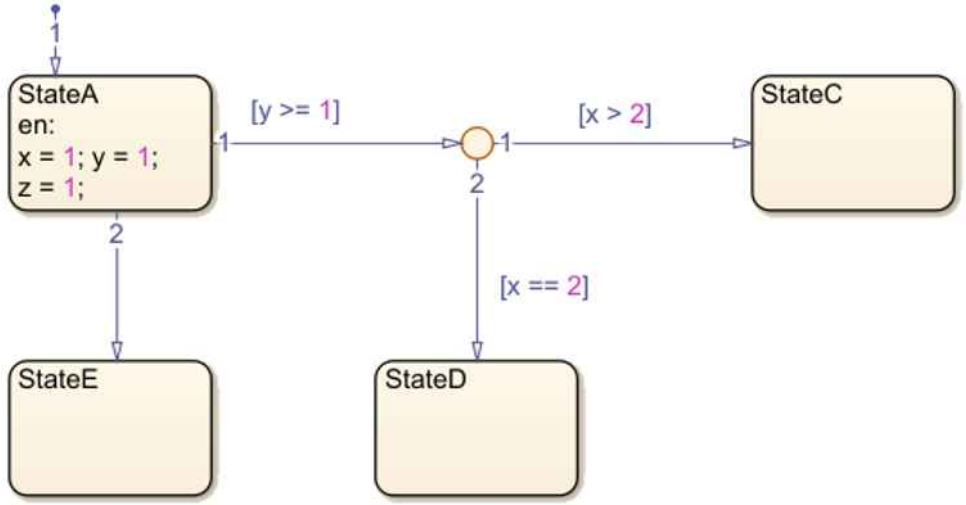
1. Out of StateA
    1. Transition 1 (pass)
    2. Junction
        1. Junction Transition 1 (fail)
        2. Junction Transition 2 (pass)

- The Stateflow chart is initialized and the entry actions are performed for StateA
- A new time step occurs and the chart wakes up
- At this time step x = 1, y = 1, and z = 1
- Transition 1 from StateA is marked for evaluation
- Transition 1 from StateA has a condition
- The condition is true
- The destination of transition 1 from StateA is not a state (it is a junction)
- But, the junction does have outgoing transitions
- Transition 1 from the junction is marked for evaluation
- Transition 1 from the junction has a condition
- The condition is false
- Transition 2 from the junction is marked for evaluation
- Transition 2 from the junction does not have a condition
- The destination of transition 2 from the junction is a state (StateD)
- StateD is marked for entry, and StateA is marked for exit

Outgoing Transition with Backtracking

- When all outgoing transitions from a source are invalid or do not end with a terminating junction, but there are previously unevaluated transitions, Stateflow returns to the previous state or junction to evaluate all possible paths

Outgoing Transition with Backtracking



Transition Evaluation Order

1. Out of StateA
   1. Transition 1 (pass)
   2. Junction
      1. Junction Transition 1 (fail)
      2. Junction Transition 2 (fail)
   3. Transition 2 (pass)

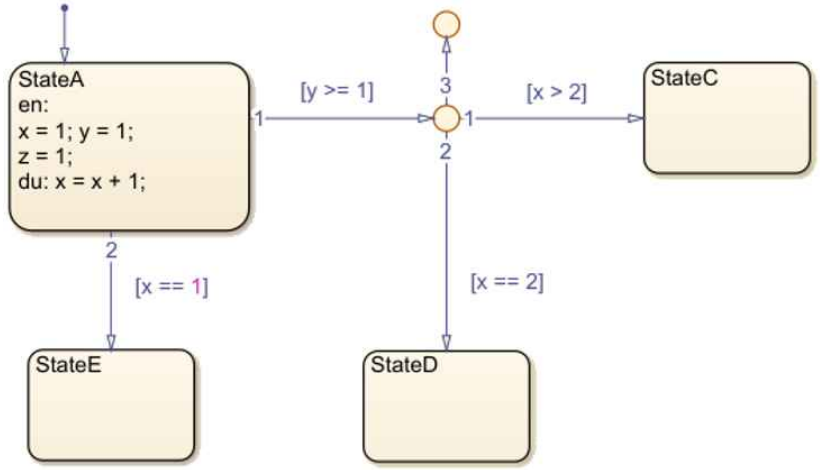
Backtracking →

- The Stateflow chart is initialized and the entry actions are performed for StateA
- A new time step occurs and the chart wakes up
- At this time step x = 1, y = 1, and z = 1
- Transition 1 from StateA is marked for evaluation
- Transition 1 from StateA has a condition
- The condition is true
- The destination of transition 1 from StateA is not a state
- The junction does have outgoing transitions
- Transition 1 from the junction is marked for evaluation
- Transition 1 from the junction has a condition
- The condition is false
- Transition 2 from the junction is marked for evaluation
- Transition 2 from the junction has a condition
- The condition is false
- At this point, Stateflow backtracks to evaluagte Transition 2 from State A
- Transition 2 from StateA is marked for evaluation
- Transition 2 from StateA does not have a condition
- The destination of transition 2 from StateA is a state (StateE)
- StateE is marked for entry, and StateA is marked for exit

Preventing Backtracking in an Outgoing Transition

- A terminating junction can be added in certain situations to prevent backtracking

Outgoing Transition with Backtracking



Transition Evaluation Order

1. Out of StateA
   1. Transition 1 (pass)
   2. Junction
      1. Junction Transition 1 (fail)
      2. Junction Transition 2 (fail)
      3. Junction Transition 3 (fail)

Backtracking Prevented

- The Stateflow chart is initialized and the entry actions are performed for StateA
- A new time step occurs and the chart wakes up
- At this time step x = 1, y = 1, and z = 1
- Transition 1 from StateA is marked for evaluation
- Transition 1 from StateA has a condition
- The condition is true
- The destination of transition 1 from StateA is not a state
- The junction does have outgoing transitions
- Transition 1 from the junction is marked for evaluation
- Transition 1 from the junction has a condition
- The condition is false
- Transition 2 from the junction is marked for evaluation
- Transition 2 from the junction has a condition
- The condition is false
- Transition 3 from the junction is marked for evaluation
- Transition 3 from the junction does not have a condition
- The destination is not a state and does not have any outgoing transitions
- No transition takes place, the system remains in StateA and the next time step occurs

Condition Actions and Transition Actions

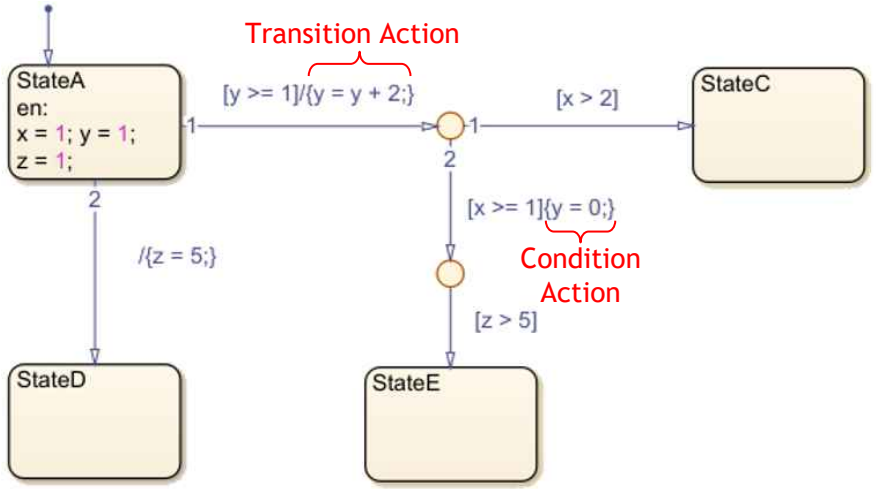
Condition Actions

- In transition label syntax, condition actions follow the transition condition and are enclosed in curly braces ({ })
- Condition actions are executed when the condition is evaluated as true but before the transition path has been determined to be valid

Transition Actions

- In transition label syntax, transition actions are preceded with a forward slash (/) and are enclosed in curly braces ({ })
- Transition actions execute only after the transition path is determined to be valid

Condition Actions and Transition Actions



Transition Evaluation Order

1. Out of StateA
    1. Transition 1 (pass)
    2. Junction
        1. Junction Transition 1 (fail)
        2. Junction Transition 2 (pass)
            1. Junction (fail)
    3. Transition 2 (pass)

- At this time, x = 1, y = 1, z = 1
- Transition 1 from StateA is marked for evaluation
- Transition 1 from StateA has a condition ([y >= 1])
- The condition is true
- There are no condition actions
- The destination of transition 1 from StateA is not a state
- But, the junction has outgoing transitions
- Transition 1 from the junction is marked for evaluation
- Transition 1 from the junction has a condition ([x > 2])
- The condition is false
- Transition 2 from the junction is marked for evaluation
- Transition 2 from the junction has a condition ([x >= 1])
- The condition is true
- There is a condition action ({y = 0;}
- Now y = 0
- The junction has outgoing transitions
- The transition from the junction is marked for evaluation
- Transition 1 from the junction has a condition ([z >= 5])
- The condition is false
- Transition 2 from StateA is marked for evaluation
- Transition 2 from StateA does not have a condition
- The destination of transition 2 from StateA is a state (StateD)
- StateD is marked for entry, and StateA is marked for exit
- Execute the transition action for this valid path (/{z = 5})
- Now z = 5

Section 7

References

References

1. The Mathworks (2021). *Stateflow Getting Started Guide.* Retrieved from https://www.mathworks.com/help/pdf_doc/stateflow/stateflow_gs.pdf
2. The Mathworks (2021). *Stateflow User's Guide.* Retrieved from https://www.mathworks.com/help/pdf_doc/stateflow/stateflow_ug.pdf
3. The Mathworks (2021). *MATLAB.* Retrieved from https://se.mathworks.com/products/matlab.html
4. The Mathworks (2021). *MATLAB Plot Gallery.* Retrieved from https://se.mathworks.com/products/matlab/plot-gallery.html
5. The Mathworks (2021). *Simulink.* Retrieved from https://www.mathworks.com/products/simulink.html
6. The Mathworks (2021). *States.* Retrieved from https://www.mathworks.com/help/stateflow/ug/states.html
7. The Mathworks (2021). *Enter a Chart or State.* Retrieved from https://www.mathworks.com/help/stateflow/ug/chart-initialization-and-entry-actions.html
8. The Mathworks (2021). *Exit a State.* Retrieved from https://www.mathworks.com/help/stateflow/ug/chart-exit-actions.html
9. The Mathworks (2021). *Evaluate Transitions.* Retrieved from https://www.mathworks.com/help/stateflow/ug/evaluate-transitions.html
10. The Mathworks (2021). *Execution of a Stateflow Chart.* Retrieved from https://www.mathworks.com/help/stateflow/ug/chart-during-actions.html
11. The Mathworks (2022). *Stateflow Semantics.* Retrieved from https://www.mathworks.com/help/stateflow/ug/what-do-semantics-mean-for-stateflow-charts.html
12. Wikipedia article "Simulink". (2022). https://en.wikipedia.org/wiki/Simulink
13. Wikipedia article "Stateflow". (2022). https://en.wikipedia.org/wiki/Stateflow
14. Soliman, W., Patel, V. (2015). *An Efficient Synchronization Control Scheme for Triaxial Motion System.* Retrieved from https://www.researchgate.net/publication/318597616_An_Efficient_Synchronization_Control_Scheme_for_Triaxial_Motion_System/figures?lo=1