Cyber-Physical Systems

Discrete Dynamics

IECE 553/453– Fall 2022

Prof. Dola Saha

Discrete Systems

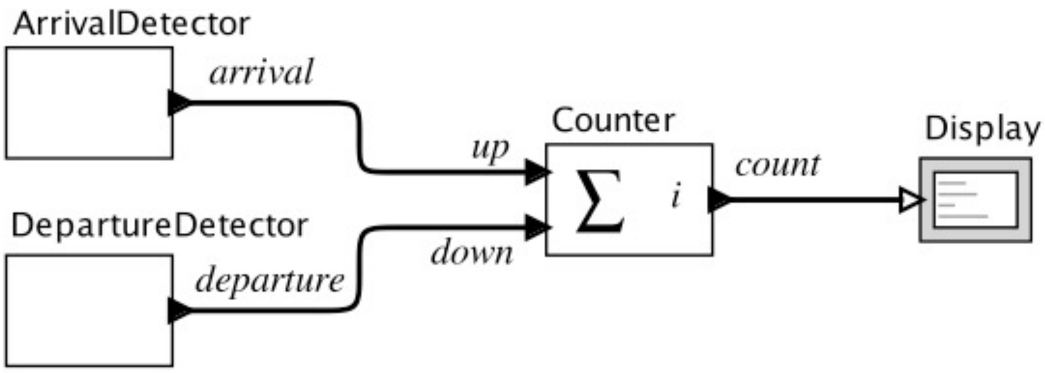
- **Discrete** = “individually separate / distinct”
- A **discrete system** is one that operates in a sequence of discrete *steps* or has signals taking discrete *values*.
- It is said to have **discrete dynamics**.
- A discrete event occurs at an instant of time rather than over time.

Discrete Systems: Example

- Count the number of cars that are present in a parking garage by sensing cars enter and leave the garage. Show this count on a display.

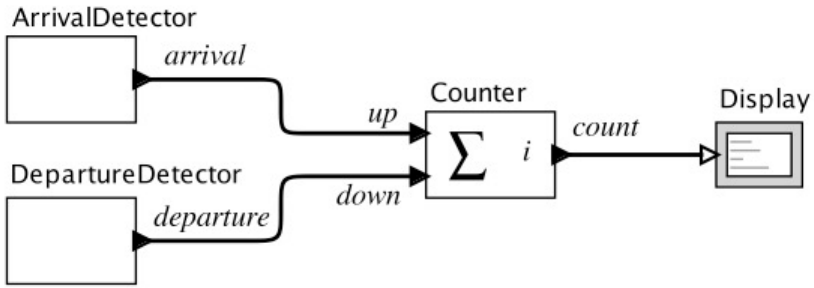
Discrete Systems

- Example: count the number of cars in a parking garage by sensing those that enter and leave:



Discrete Systems

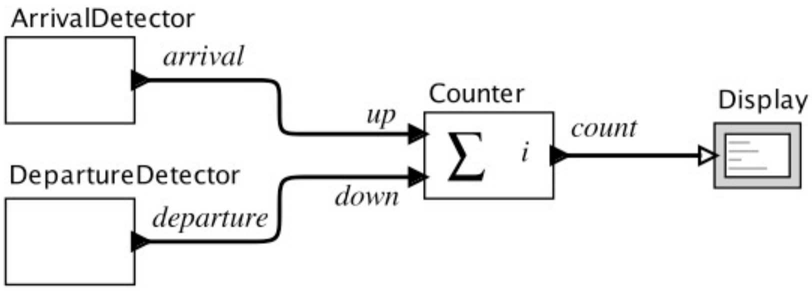
- Example: count the number of cars that enter and leave a parking garage:



- Pure signal: $up : \mathbb{R} \rightarrow \{absent, present\}$
  - Carries no value, information is being present or absent
- at any time t ∈ R, the input up(t) is
  - either *absent*, meaning that there is no event at that time,
  - or *present*, meaning that there is.

Discrete Systems

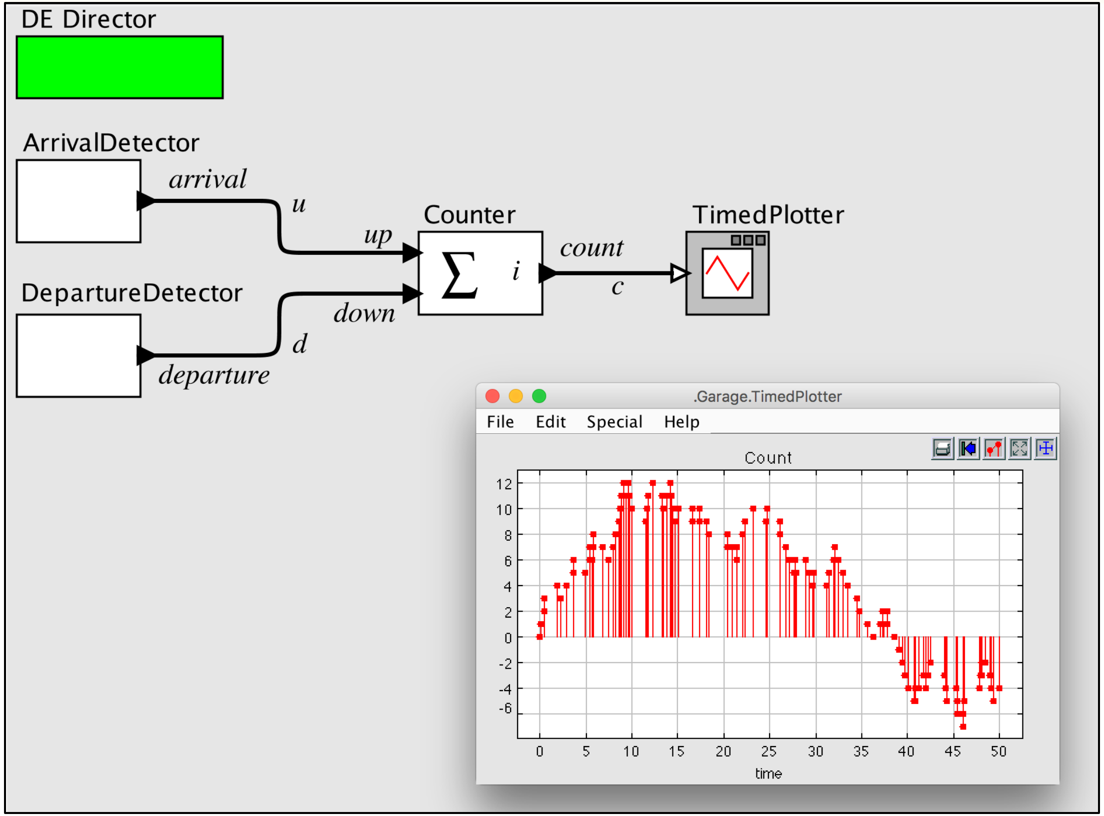
- Example: count the number of cars that enter and leave a parking garage:



- Pure signal: $up: \mathbb{R} \rightarrow \{absent, present\}$
- Discrete actor: $Counter: (\mathbb{R} \rightarrow \{absent, present\})^P \rightarrow (\mathbb{R} \rightarrow \{absent\} \cup \mathbb{N})$

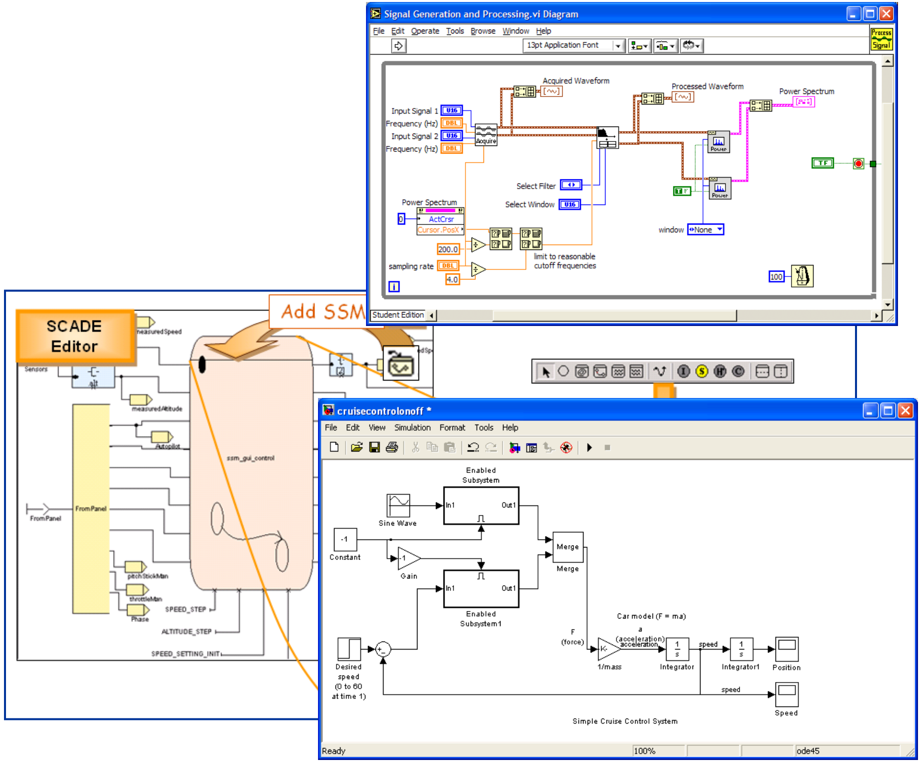
  $P = \{up, down\}$

Demonstration of Ptolemy II Model



Actor Modeling Languages

- LabVIEW
- Simulink
- Scade
- …
- Reactors
- StreamIT
- …

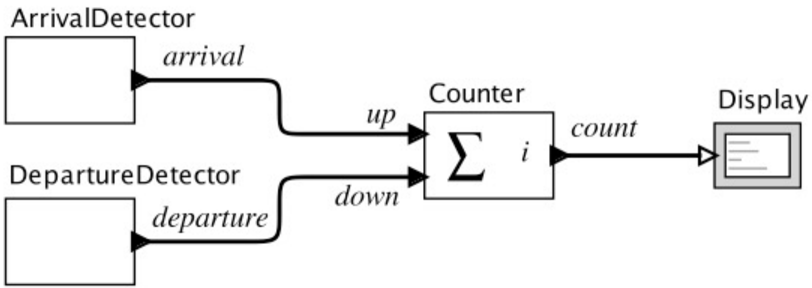


Reaction / Transition

For any $t \in \mathbb{R}$ where $up(t) \neq absent$ or $down(t) \neq absent$ the Counter **reacts**. It produces an output value in $\mathbb{N}$ and changes its internal **state**.

State: condition of the system at a particular point in time

- Encodes everything about the past that influences the system's reaction to current input



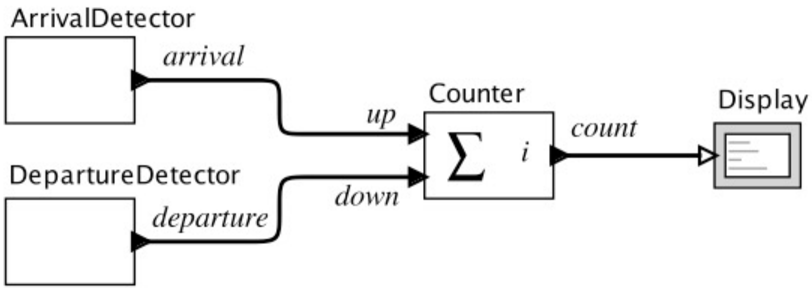
Inputs and Outputs at a Reaction

For $t \in \mathbb{R}$ the inputs are in a set

$$Inputs = (\{\mathit{up}, \mathit{down}\} \to \{\, absent, present\})$$

and the outputs are in a set

$$Outputs = (\{count\} \to \{\, absent\} \cup \mathbb{N})\ ,$$



Question

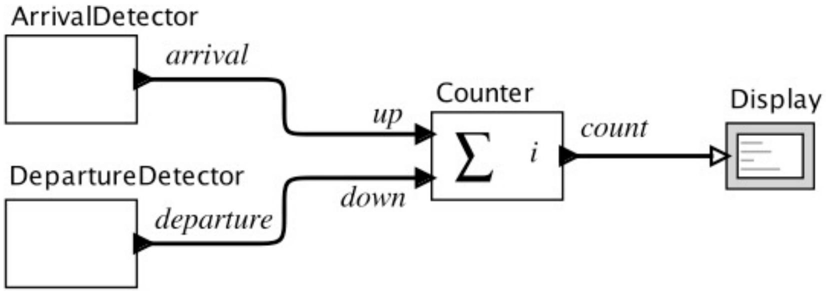
- What are some scenarios that the given parking garage (interface) design does not handle well?

For $t \in \mathbb{R}$ the inputs are in a set

$$Inputs = (\{\mathsf{up}, \mathsf{down}\} \rightarrow \{absent, present\})$$

and the outputs are in a set

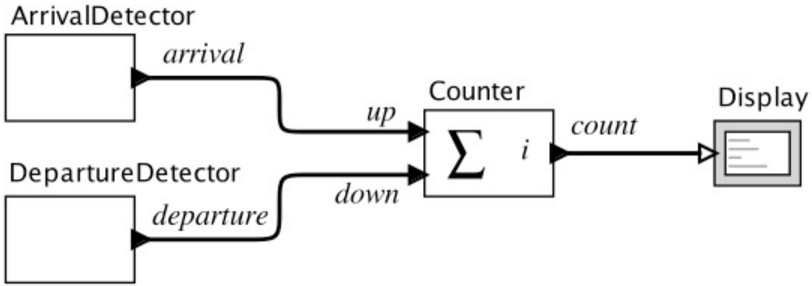
$$Outputs = (\{count\} \rightarrow \{absent\} \cup \mathbb{N}) \ ,$$



State Space

A practical parking garage has a finite number M of spaces, so the state space for the counter is

$$States = \{0, 1, 2, \cdots, M\} \ .$$



Finite State Machine (FSM)

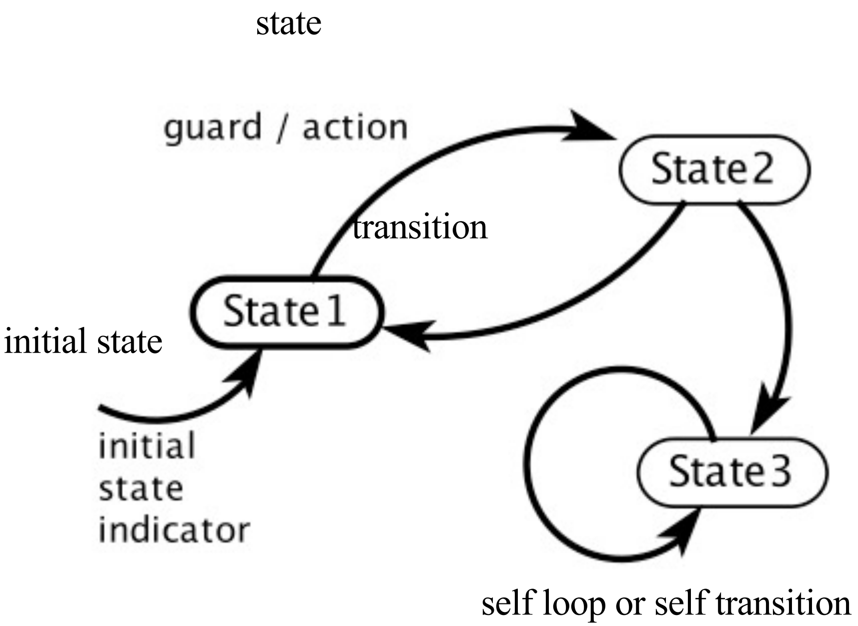
- A state machine is a model of a system with discrete dynamics
  - at each reaction maps inputs to outputs
  - Map may depend on current state
- An FSM is a state machine where the set *States* is finite. $States = \{\text{State1}, \text{State2}, \text{State3}\}$

FSM Notation

Input declarations, Output declarations, Extended state declarations

The guard determines whether the transition may be taken on a reaction.

The action specifies what outputs are produced on each reaction.



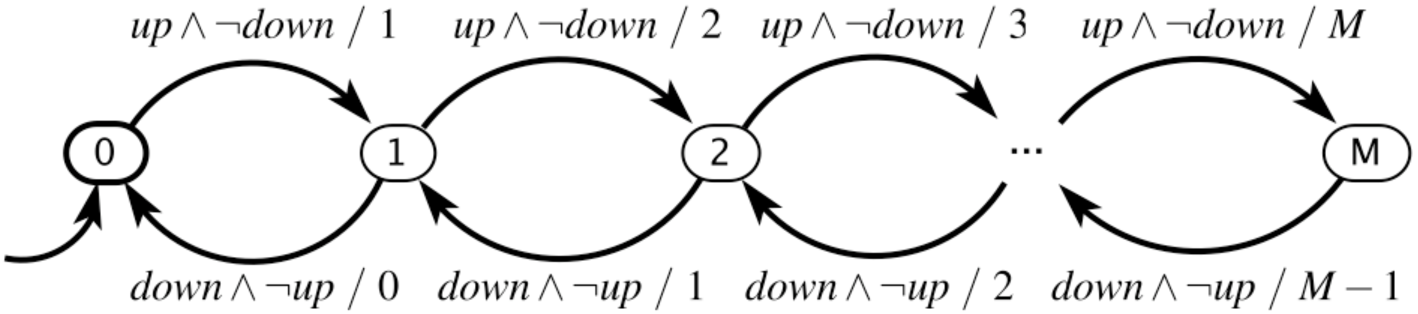
Examples of Guards for Pure Signals

true	Transition is always enabled.
p_1	Transition is enabled if p_1 is *present*.
$\neg p_1$	Transition is enabled if p_1 is *absent*.
$p_1 \wedge p_2$	Transition is enabled if both p_1 and p_2 are *present*.
$p_1 \vee p_2$	Transition is enabled if either p_1 or p_2 is *present*.
$p_1 \wedge \neg p_2$	Transition is enabled if p_1 is *present* and p_2 is *absent*.

Guards for Signals

p_3	Transition is enabled if p_3 is *present* (not *absent*).
$p_3 = 1$	Transition is enabled if p_3 is *present* and has value 1.
$p_3 = 1 \wedge p_1$	Transition is enabled if p_3 has value 1 and p_1 is *present*.
$p_3 > 5$	Transition is enabled if p_3 is *present* with value greater than 5.

Garage Counter FSM



Guard $g \subseteq Inputs$ is specified using the shorthand

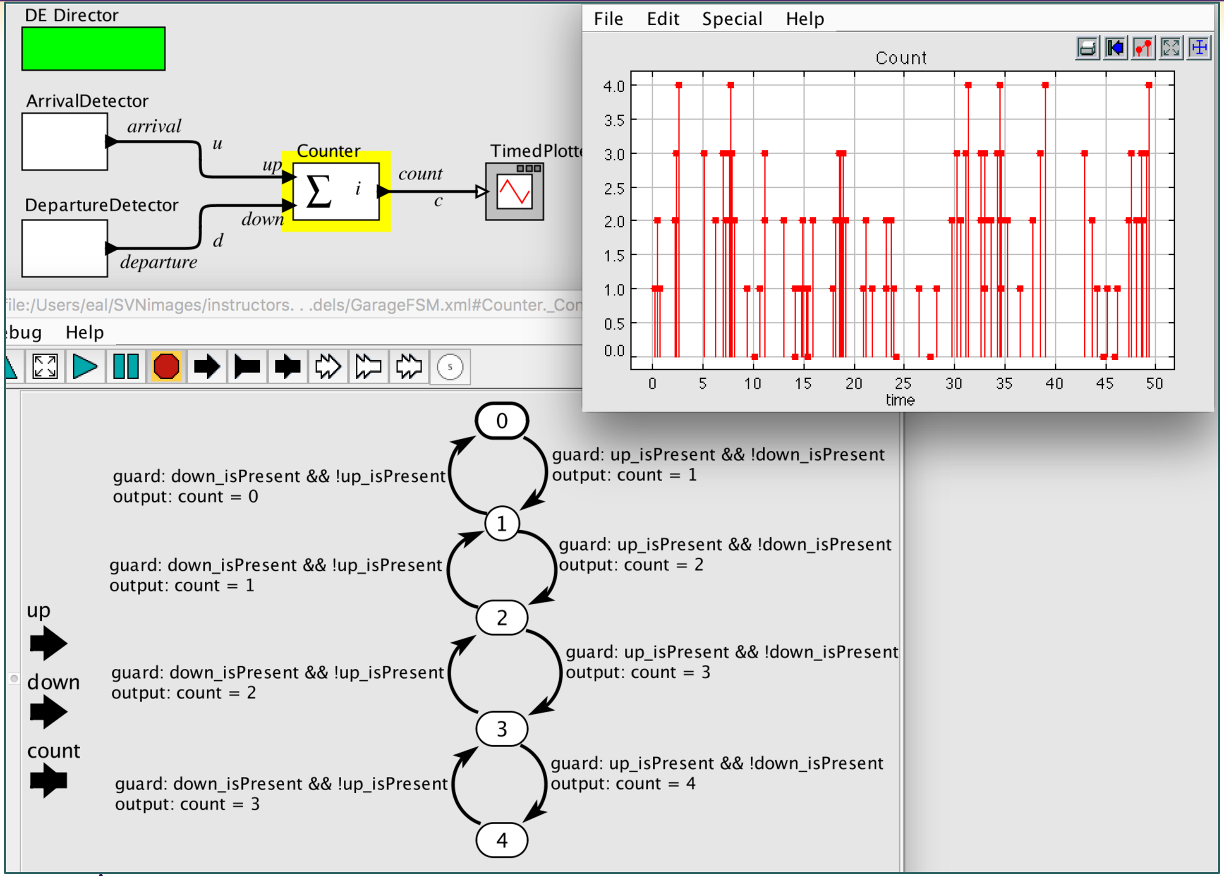
$$up \wedge \neg down$$

which means

$$g = \{\{up\}\} \ .$$

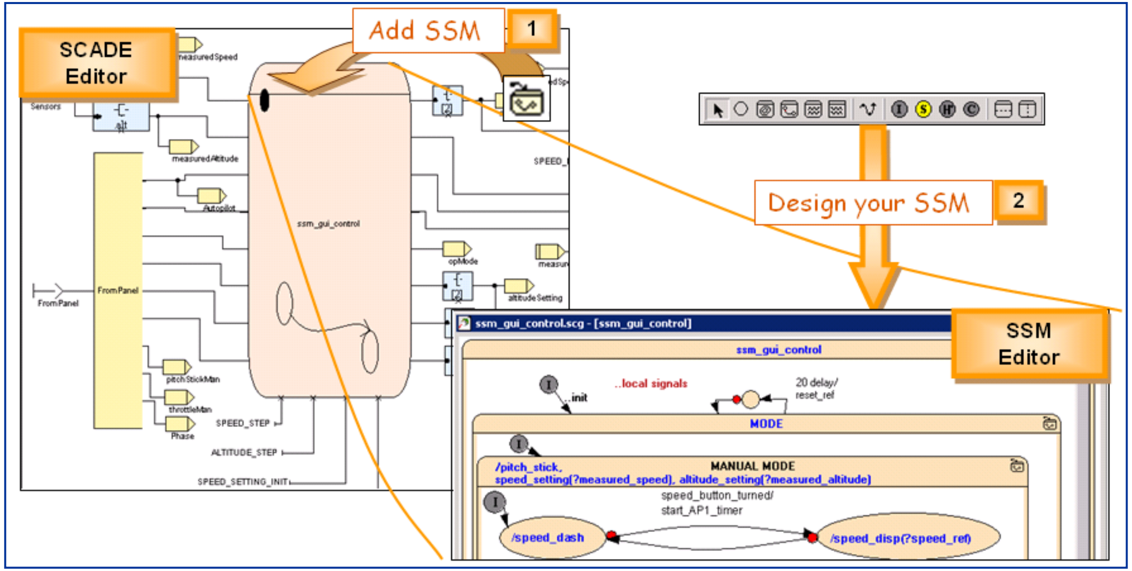
Inputs(up) = present and Inputs(down) = absent

Ptolemy II Model

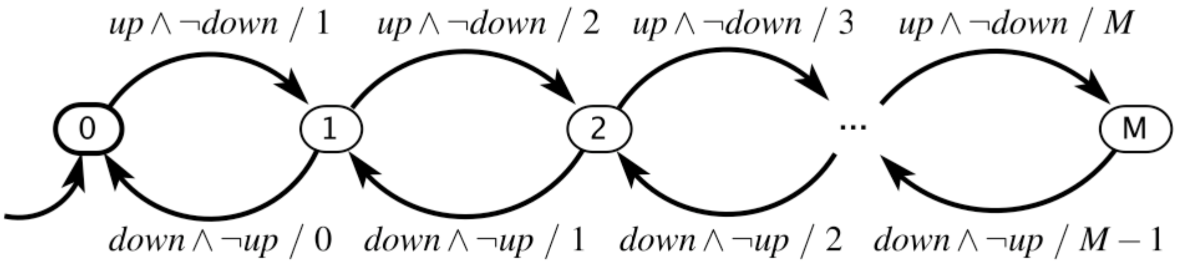


FSM Modeling Languages / Frameworks

- LabVIEW Statecharts
- Simulink Stateflow
- Scade
- …



Garage Counter Mathematical Model



Formally: $(\mathit{States}, \mathit{Inputs}, \mathit{Outputs}, \mathit{update}, \mathit{initialState})$, where

- $\mathit{States} = \{0, 1, \cdots, M\}$
- $\mathit{Inputs} = (\{\text{up}, \text{down}\} \rightarrow \{\mathit{absent}, \mathit{present}\}$
- $\mathit{Outputs} = (\{\mathit{count}\} \rightarrow \{\mathit{absent}\} \cup \mathbb{N})$
- $\mathit{update} : \mathit{States} \times \mathit{Inputs} \rightarrow \mathit{States} \times \mathit{Outputs}$
- $\mathit{initialState} = 0$

Transition Function

$$(s(n+1), y(n)) = \mathit{update}(s(n), x(n))$$

The update function is given by

$$\mathit{update}(s, i) = \begin{cases} (s+1, s+1) & \text{if } s < M \\ & \wedge\, i(\mathit{up}) = \mathit{present} \\ & \wedge\, i(\mathit{down}) = \mathit{absent} \\ (s-1, s-1) & \text{if } s > 0 \\ & \wedge\, i(\mathit{up}) = \mathit{absent} \\ & \wedge\, i(\mathit{down}) = \mathit{present} \\ (s, \mathit{absent}) & \text{otherwise} \end{cases}$$

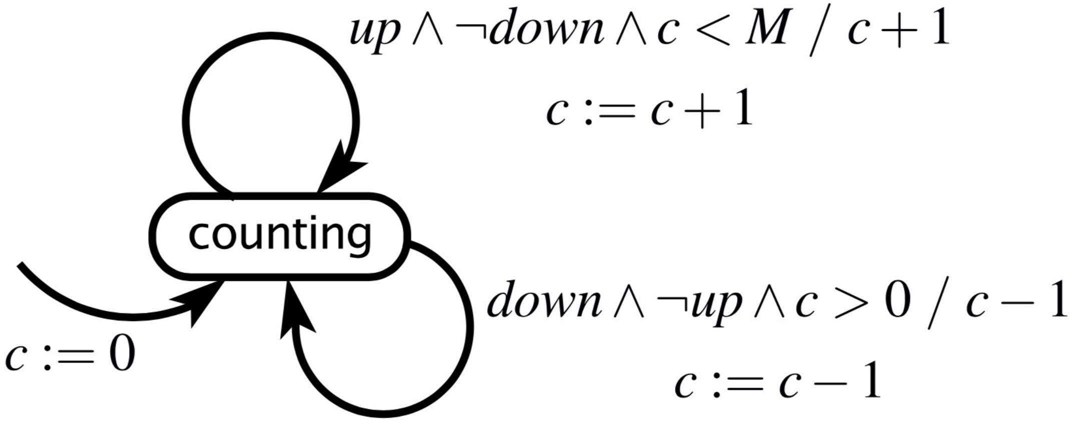
FSM: Definitions

- Stuttering: (possibly implicit) default transition that is enabled
  - when inputs are absent it does not change state and produces absent outputs.
- Deterministic (given the same inputs it will always produce the same outputs)
  - if, for each state, there is at most one transition enabled by each input value.
  - formal definition of an FSM ensures that it is deterministic, since *update* is a function.
- Receptive (ensures that a state machine is always ready to react to any input, and does not "get stuck" in any state)
  - if, for each state, there is at least one transition possible on each input symbol.
  - formal definition of an FSM ensures that it is receptive, since *update* is a function, not a partial function.

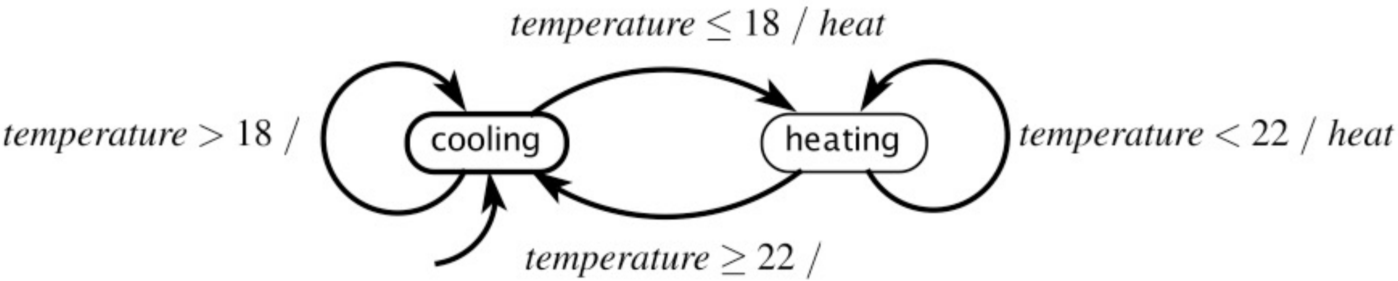
Extended State Machine

- augments the FSM model with ***variables*** that may be read and written as part of taking a transition between states

variable: $c \colon \{0, \cdots, M\}$
inputs: $up, down \colon$ pure
output: $count \colon \{0, \cdots, M\}$



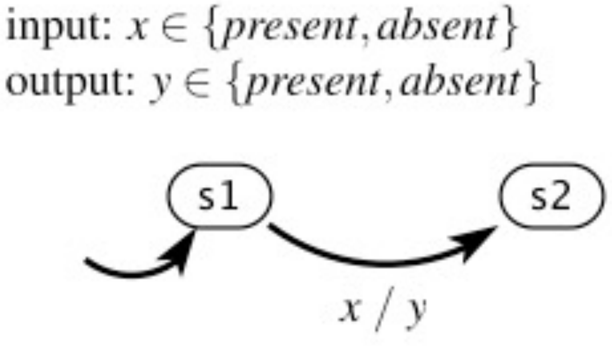
Example of Thermostat



When does a reaction occur?

- Suppose all inputs are discrete and a reaction occurs *when any input is present*. Then the below transition will be taken whenever the current state is s1 and *x* is present.

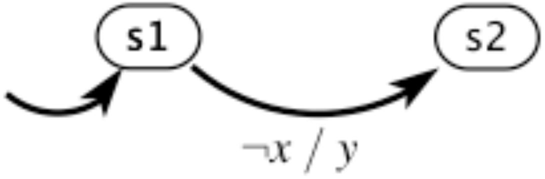
- This is an *event*



When does a reaction occur?

- Suppose *x* and *y* are discrete and pure signals. When does the transition occur?

input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$

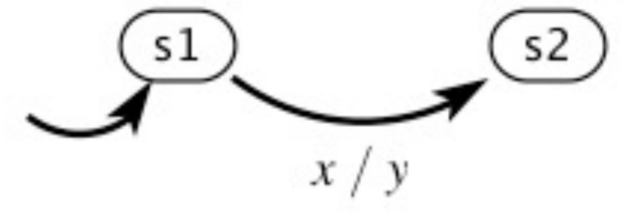


Answer: when the *environment* triggers a reaction and x is absent.
If this is a (complete) event-triggered model, then the transition will never be taken because the reaction will only occur when x is present!

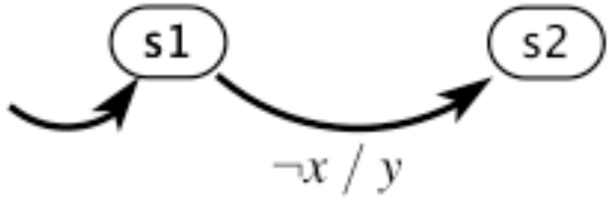
When does a reaction occur?

- Suppose all inputs are discrete and a reaction occurs *on the tick of an external clock*.
- This is a *time-triggered model*.

input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$



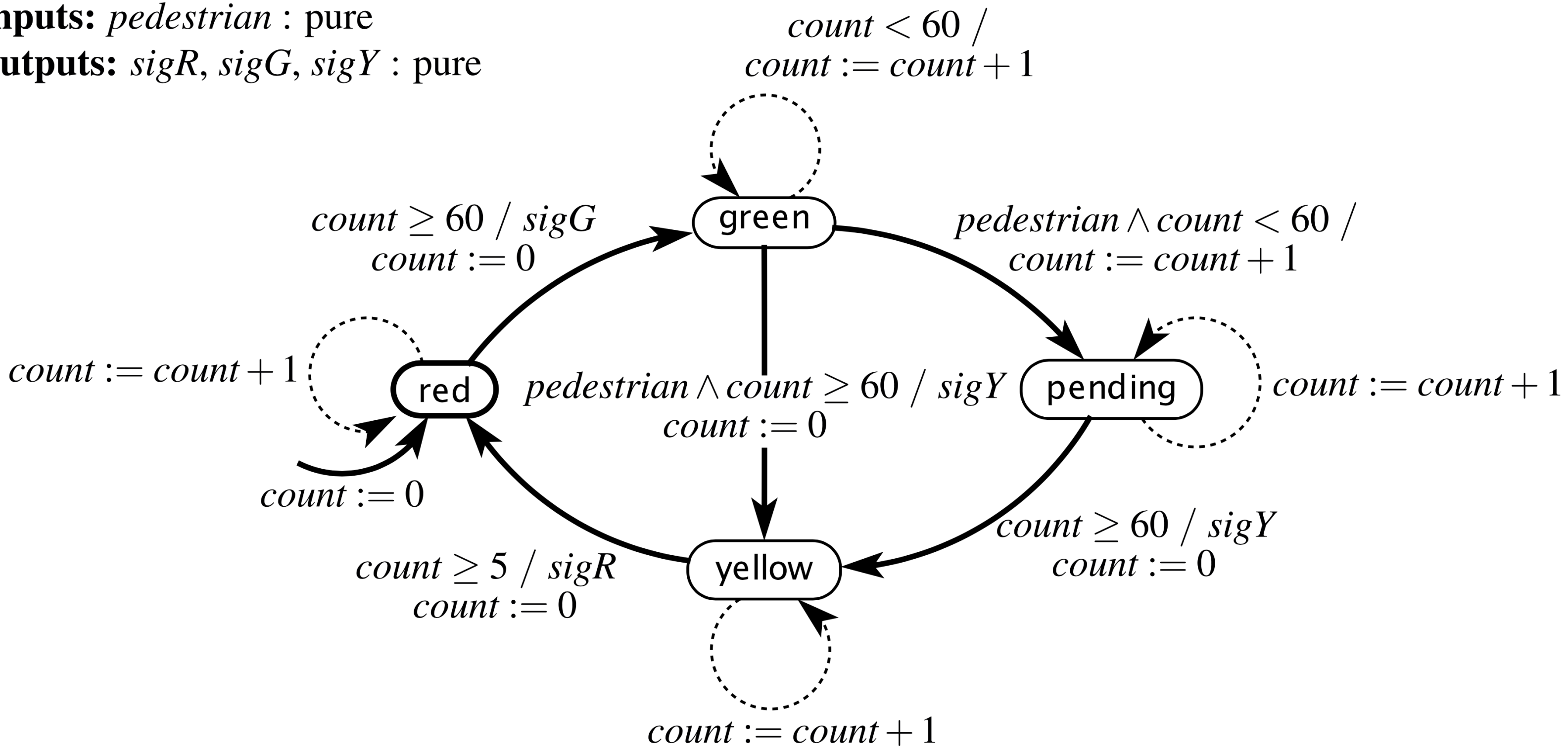
input: $x \in \{present, absent\}$
output: $y \in \{present, absent\}$



Extended FSM of traffic light

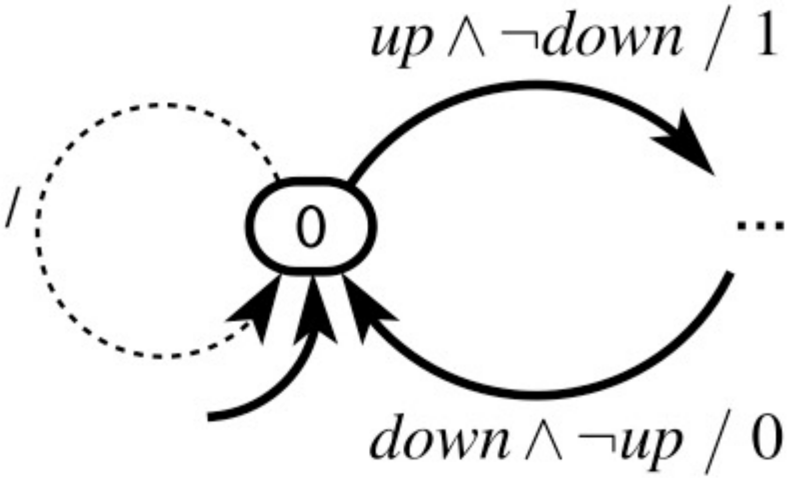
Reacts at regular intervals (1 sec)

variable: *count*: $\{0, \cdots, 60\}$
inputs: *pedestrian* : pure
outputs: *sigR*, *sigG*, *sigY* : pure



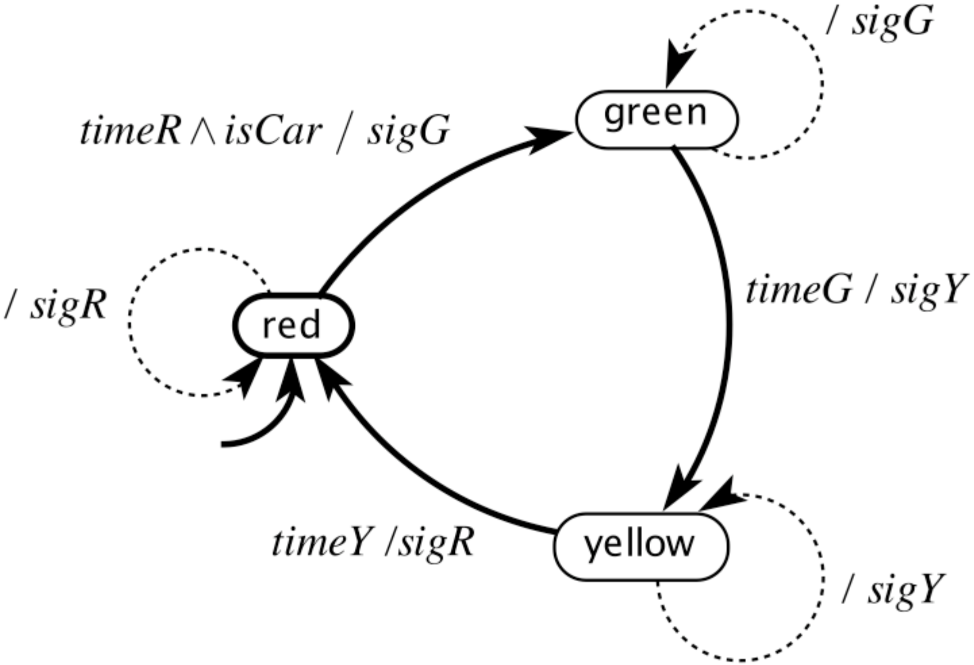
More Notation: Default Transitions

- A default transition is enabled if it either has no guard or the guard evaluates to true. When is the below default transition enabled?



Default Transitions

➢ Example: Traffic Light Controller



FSM State Transition Table

State-transition table

(S: state, I: input, O: output)

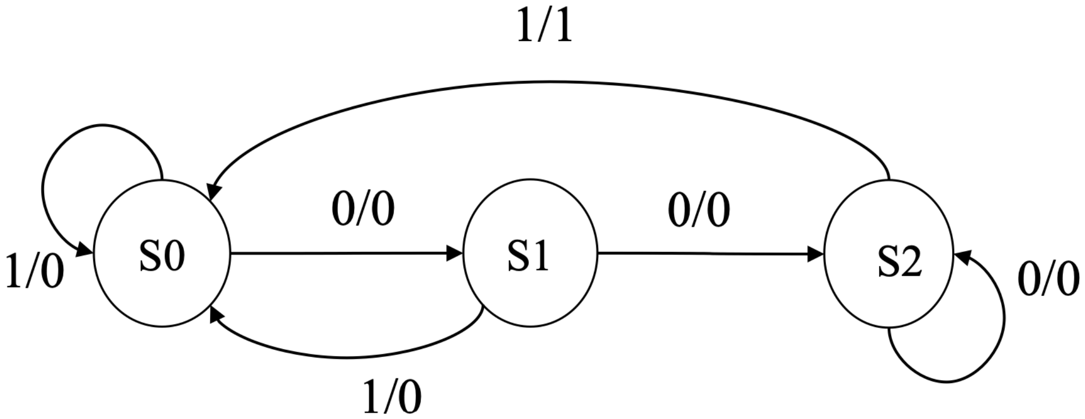
Current state \ Input	I_1	I_2	…	I_n
S_1	S_i/O_x	S_j/O_y	…	S_k/O_z
S_2	$S_{i'}/O_{x'}$	$S_{j'}/O_{y'}$	…	$S_{k'}/O_{z'}$
…	…	…	…	…
S_m	$S_{i''}/O_{x''}$	$S_{j''}/O_{z''}$	…	$S_{k''}/O_{z''}$

State-transition table

(S: state, I: input, O: output, —: illegal)

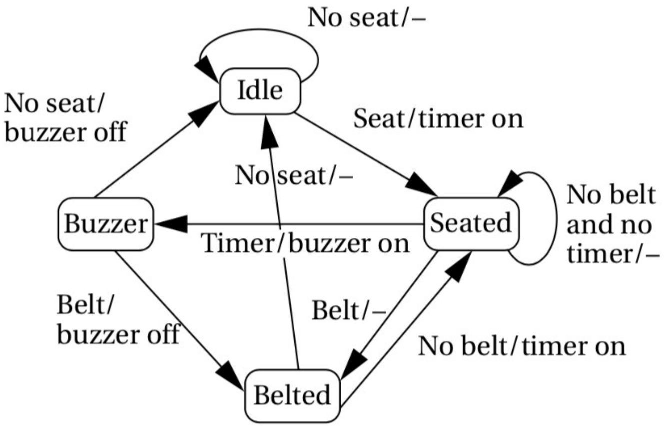
Current state \ Next state	S_1	S_2	…	S_m
S_1	I_i/O_x	—	…	—
S_2	—	—	…	I_j/O_y
…	…	…	…	…
S_m	—	I_k/O_z	…	—

Example FSM: Detect “001”



S(t)\x	0	1
S0	S1,0	S0,0
S1	S2,0	S0,0
S2	S2,0	S0,1

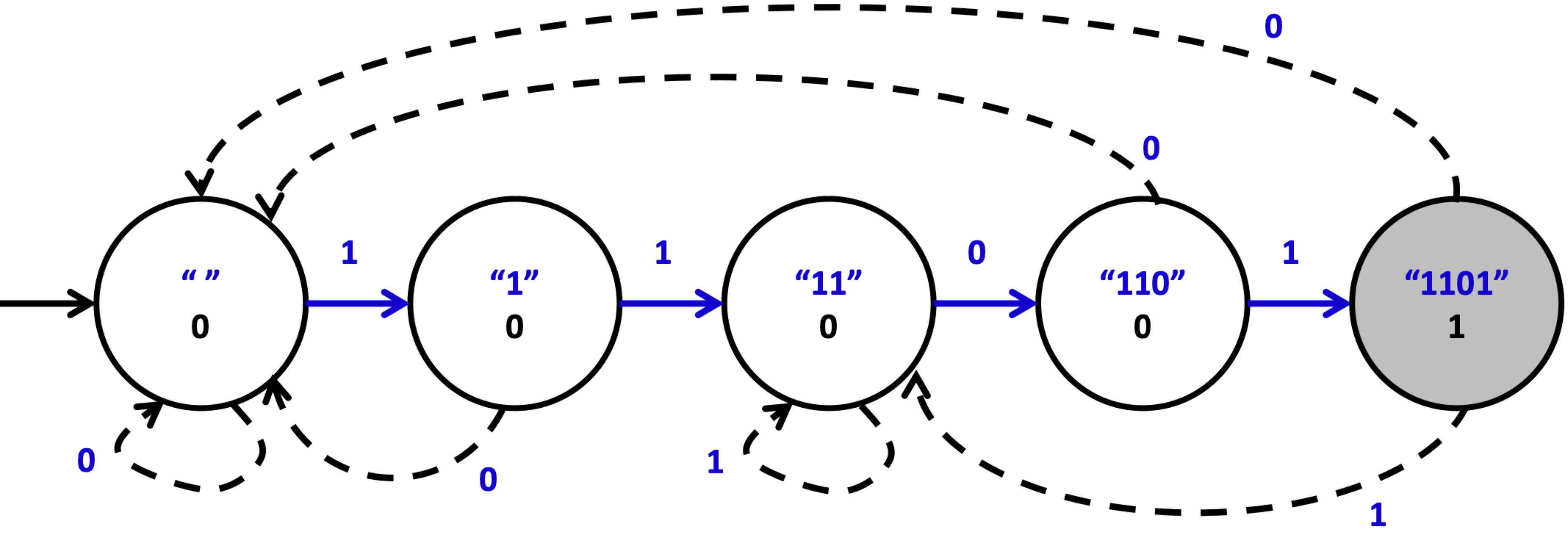
FSM to Program




```c
#define IDLE 0
#define SEATED 1
#define BELTED 2
#define BUZZER 3

switch (state) { /* check the current state */
	case IDLE:
		if (seat) { state = SEATED; timer_on = TRUE; }
		/* default case is self-loop */
		break;
	case SEATED:
		if (belt) state = BELTED; /* won't hear the
					buzzer */
		else if (timer) state = BUZZER; /* didn't put on
					belt in time */
		/* default is self-loop */
		break;
	case BELTED:
		if (!seat) state = IDLE; /* person left */
		else if (!belt) state = SEATED; /* person still
					in seat */
		break;
	case BUZZER:
		if (belt) state = BELTED; /* belt is on—turn off
					buzzer */
		else if (!seat) state = IDLE; /* no one in
					seat—turn off buzzer */
		break;
}
```

Example FSM: Recognize "1101"



FSM that accepts “UAlbany”

FSM that detects “CAT” or “DOG”

FSM that detects “CYBER”