A four-way intersection has red/green traffic lights that are controlled with timers.

Traffic can only move in one direction at a time: NS or EW.
A four-way intersection has red/green traffic lights that are controlled with timers.

Traffic can only move in one direction at a time: NS or EW.
A four-way intersection has red/green traffic lights that are controlled with timers.

Traffic can only move in one direction at a time: NS or EW.

We can show this graphically with a state diagram.
A Finite-State Machine (FSM) is a model of the discrete dynamics of a system that has a finite number of discrete **states**. Transitions between states are caused by **events**, such as:

- the expiration of a timer
- a change in a sensor

---

**State Diagram**

- EW green → EW red (timer expired)
- NS red → NS green (timer expired)

**State Table**

<table>
<thead>
<tr>
<th>Current State</th>
<th>Event</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>EW grn/NS rd</td>
<td>timer exp</td>
<td>EW rd/NS grn</td>
</tr>
<tr>
<td>EW rd/NS grn</td>
<td>timer exp</td>
<td>EW grn/NS rd</td>
</tr>
</tbody>
</table>
Finite-State Machines are often used to design control systems...

A garage door opening system

If the door is closed and I press the button, the door begins to move up.

When it reaches the top, the door activates a limit switch and stops.

If the door is open and I press the button, the door begins to move down.

When it reaches the bottom, the door activates another limit switch and stops.
Finite-State Machines are often used to design control systems...

A garage door opening system

- button
- garage door controller
- motor
- door
- limit switch

block diagram
Finite-State Machines are often used to design control systems...

A garage door opening system

block diagram

button

garage door controller

motor

doors

limit switch

...we want to design the controller...
Finite-State Machines are often used to design control systems...

A garage door opening system

states
- door closed
- door open
- door closing
- door opening

events
- button press
- limit switch
Finite-State Machines are often used to design control systems...
Implementing a Finite-State Machine

...in software:

while (true)
    event = GetEvent()
    if ( state == closed AND event == button_press )
        open_door()
        state = opening
    else
    if ( state == open AND event == button_press )
        close_door()
        state = closing
    else
    if ( state == opening AND event == limit_switch )
        stop_door()
        state = open
    else
    if ( state == closing AND event == limit_switch )
        stop_door()
        state = closed
Implementing a Finite-State Machine

...in software:

Initialize state

while (true)

    event = GetEvent()
    if ( state == closed AND event == button_press )
        open_door()
        state = opening
    else
        if ( state == open AND event == button_press )
            close_door()
            state = closing
        else
            if ( state == opening AND event == limit_switch )
                stop_door()
                state = open
            else
                if ( state == closing AND event == limit_switch )
                    stop_door()
                    state = closed

    this is pseudocode... a mix of natural language and typical programming language constructs without the details of languagespecific syntax
An autonomous mobile robot must navigate through a maze.

An on-line navigation problem: solving a maze from the inside.

An on-line algorithm receives its input gradually rather than all at once.

It must make decisions based on this partial input.
An Autonomous Robot

- Robot controller (software)
- Motors
- Wheels
- Range sensor
- Bump sensor

Start button

Motion of robot in maze

Maze
FSM to model robot navigation

Forward

do something!

range sensor pointed forward

obstacle: wall

dist < 400mm
FSM to model robot navigation

- Forward
- Turning right

- Dist < 400mm
- Timer expired

- Obstacle: wall
- Range sensor pointed forward
FSM to model robot navigation (the ‘right-then-left’ algo)

Forward

Turning Right 90

Turned Right

Turning Left 180

Turned Left

dist < 400mm

timer exp

dist_left >= dist_right

timer exp

dist_left < dist_right

timer exp

measure distance

do something!
FSM to model robot navigation (wall-following)

- **Obstacle:** wall
- **Bump Sensor:** pointed forward
- **Range Sensor:** pointed rightward

Diagram:
- **Forward** state
- Transition on **dist > 400mm**
- Transition on **bump detected**
- **Do something!**