深蓝学院
shenlanxueyuan.com

Autonomous Vehicle Planning and Control

Wu Ning

Session 7

Vehicle Behavior Planning

Outline

Behavior planner/Decision making

- Behavior Planner concept
- Functionality and Challenges
- Common method

Rule based approaches

- Finite State Machine
- Behavior Tree

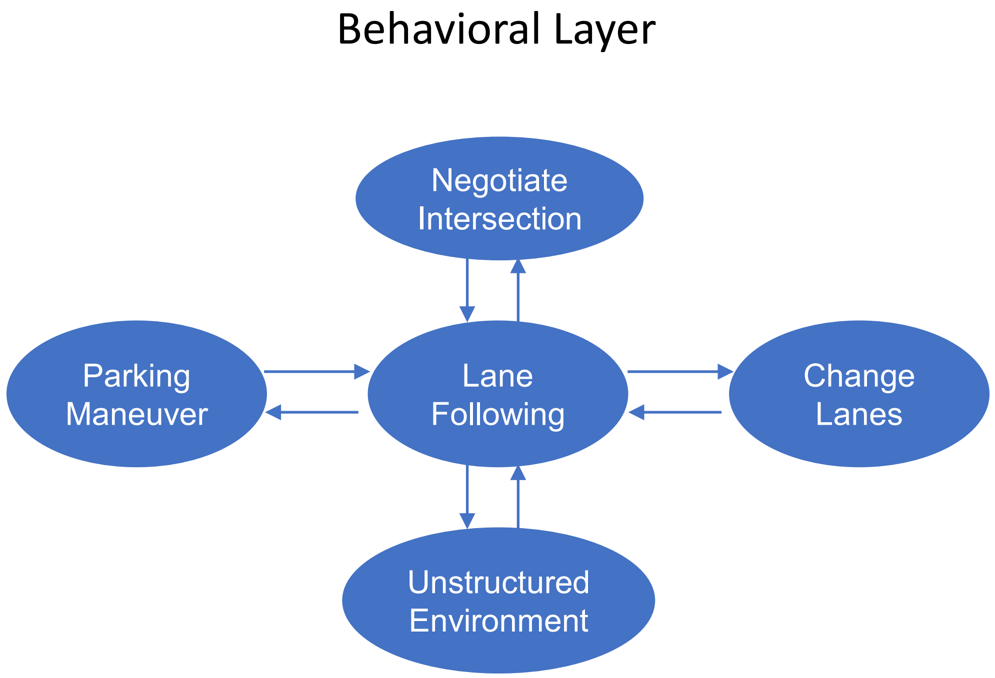
Machine Learning based approaches

- POMDP
- Pros and Cons

Behavior Planning

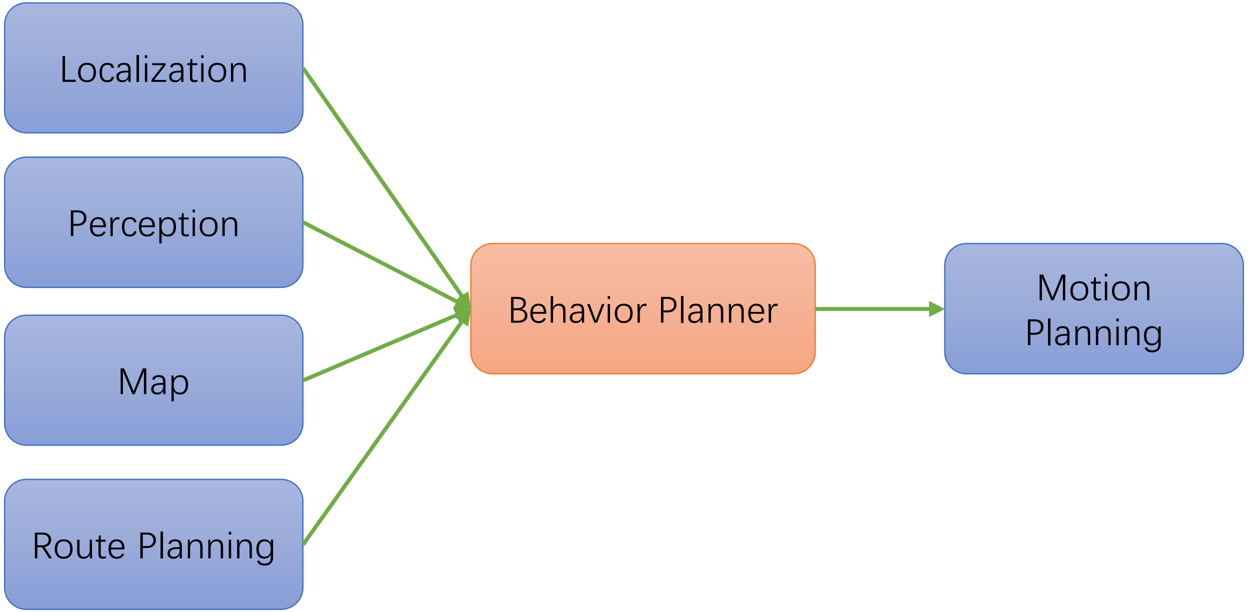
Determine high level actions:

- E.g. change lane now or later, stick to left or right lane, stop/go for crosswalk intersection



Behavior Planning

- I/O



Behavior Planning

Mechanisms of enforcing a behavior

- Limit search to subset of feasible configuration space (e.g. c-space only extends to stop sign)
- Set a local goal (e.g. current cycle plan goal to stop at stop sign)
- Set constraints for trajectory optimization (e.g. speed equal to 0 at stop sign)
- Set costs for trajectory selection (e.g. penalize trajectory running a stop sign)
- Set fake obstacles (e.g. virtual obstacle to prevent motion beyond stop sign)

How do these mechanisms affect completeness?

- *Complete with respect to constrained problem, but not searching entire workspace*

Behavior Planning

Mechanisms of enforcing a behavior

- Limit search to subset of feasible configuration space (e.g. c-space only extends to stop sign)
- Set a local goal (e.g. current cycle plan goal to stop at stop sign)
- Set constraints for trajectory optimization (e.g. speed equal to 0 at stop sign)
- Set costs for trajectory selection (e.g. penalize trajectory running a stop sign)
- Set fake obstacles (e.g. virtual obstacle to prevent motion beyond stop sign)

How do these mechanisms affect completeness?

- *Complete with respect to constrained problem, but not searching entire workspace*

What is a good behaviour planning (decision making) system

- **Good Decision Attributes:**
  - Timely
  - Account for the effect of our actions
  - Account for the actions of others
  - Reliable and repeatable

- **Good Decision Outcome**
  - Handled safely
  - Handled comfortably
  - A super rider experience

Challenges in Decision Making

Challenge 1: Decision Density

- In the decision making process, there are usually over a hundreds of agents that may interact with the vehicle. This will result in a need of more than 5000 trajectories for making a correct decision.
  - Normally decision making algorithms are running at 10-30Hz, making it a huge challenge to select the correct trajectory.

Challenge 2: Planning under uncertainty

- Even with infinitely precise models, we must account for the fact that we cannot predict the future choices of road users with perfect precision online.
  - Kinematic uncertainty
  - Existence uncertainty
  - Vehicle model uncertainty

Challenges in Decision Making

Sources of uncertainty (in expected rank order increasing size)

Localization

- Where am I now?

Control

- Where will I be later? Tracking error (lateral, longitudinal, speed, heading)

Perception

- Current world state
  - Other agent position, size, velocity, type
  - Traffic light status

Prediction

- Future world state
  - Uncertainty increases dramatically as function of time horizon

Visibility Limitation / Occupancy Likelihood

- Is there someone/something relevant that I can't see?
  - Also prediction of non-visible world evolution (other agent, traffic light)

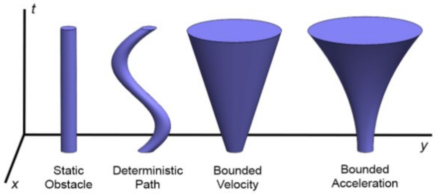


Figure 2.3: Obstacles as space-time volumes in $\Re^2 \times$ Time space, adapted from [13]. Time is shown in vertical axis. When accounting for uncertainty, obstacle size grows with respect to time due to unknown potential change in obstacle velocity.

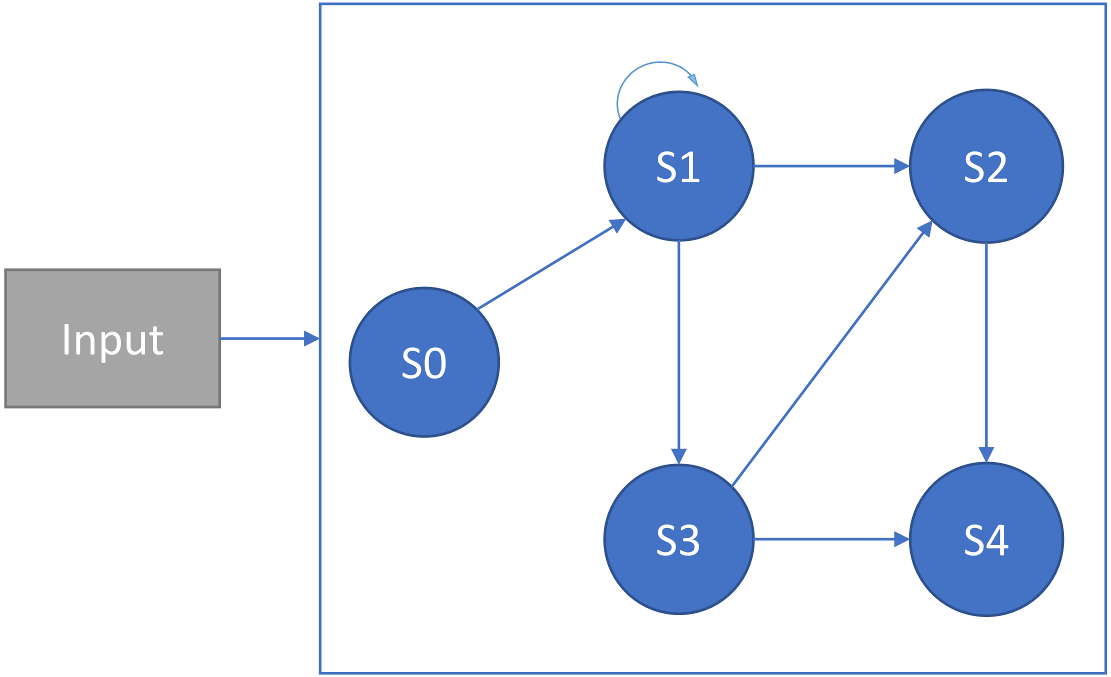
Behavior Planning

There are mainly two categories approaches:

- **Rule bases:**
  - FSM
  - Behaviour Tree
  - Minimum Violation Planning
  - Formal methods
  - etc
- **Machine Learning based:**
  - Reinforcement learning;
  - CNN
  - Deep learning
  - Decision Tree
  - etc

Finite State Machine

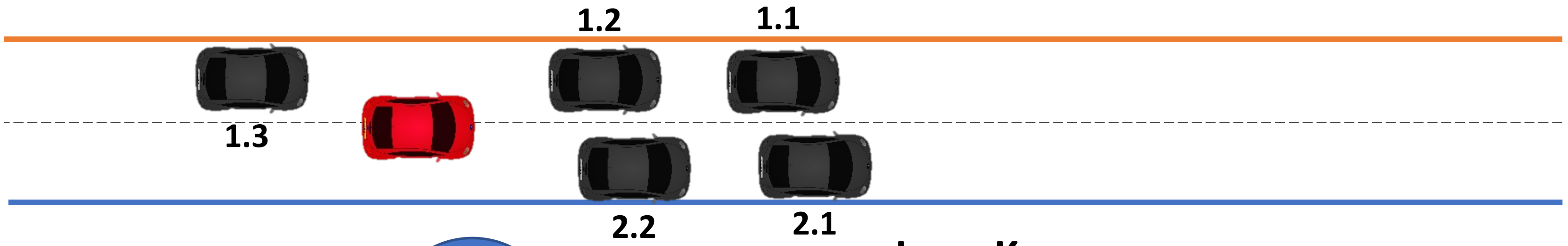
Finite State Machines

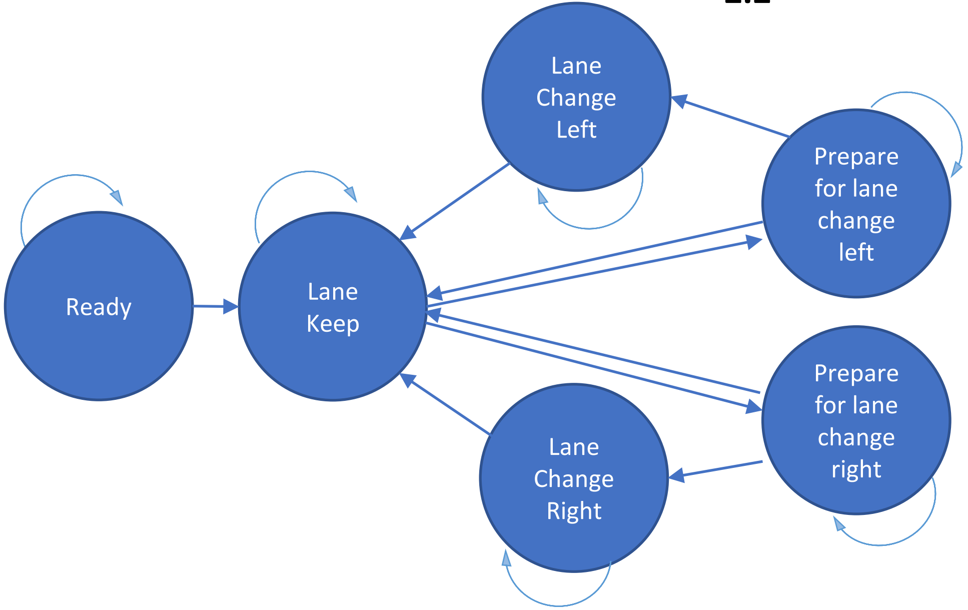


- **Accepting State:**
No transitions to other states

- **Transition Function:**
Uses input to decide what transition to make

FSM Design Diagram





Lane Keep

- d – stay near centre line of lane
- s – drive at target speed when feasible, otherwise…
    - d – stay near centre line for lane

Lane change Left/Right

- d – move left or right
- s – same rules as keep lane (for initial lane)

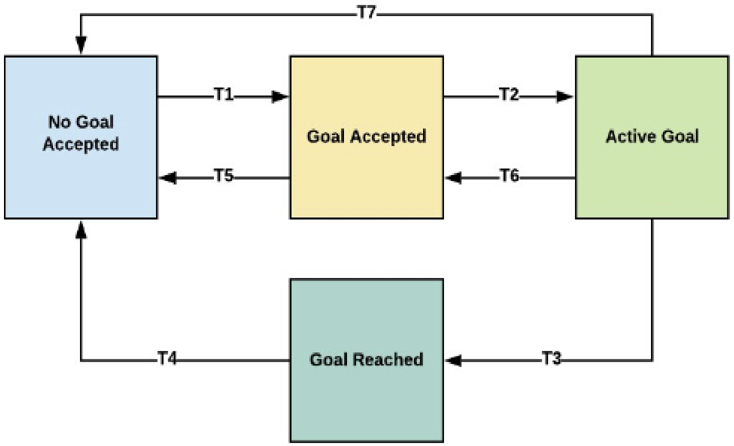
Prepare lane change Left/Right

- d – stay near centre line for current lane
- s – attempt to match position and speed of "gap" in lane
- Signal – active turning signal

Behavior Planning: Finite State Machine

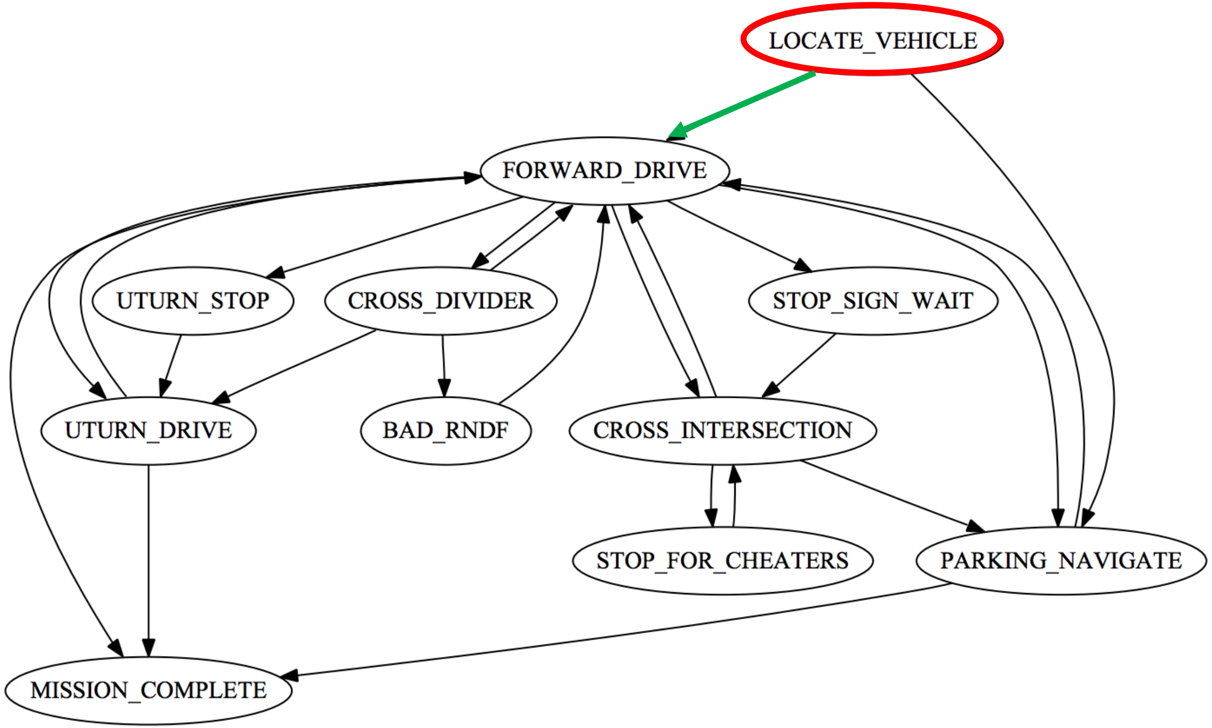
Finite State Machine to select high level “maneuver”, possibly with dedicated planner
E.g. switch to parking mode to do reverse maneuver for parking

- These may dictate local goals, or even completely different underlying algorithms
- Important to verify no undesirable properties in large and/or hierarchical FSM, eg deadlock, livelock, unreachable states



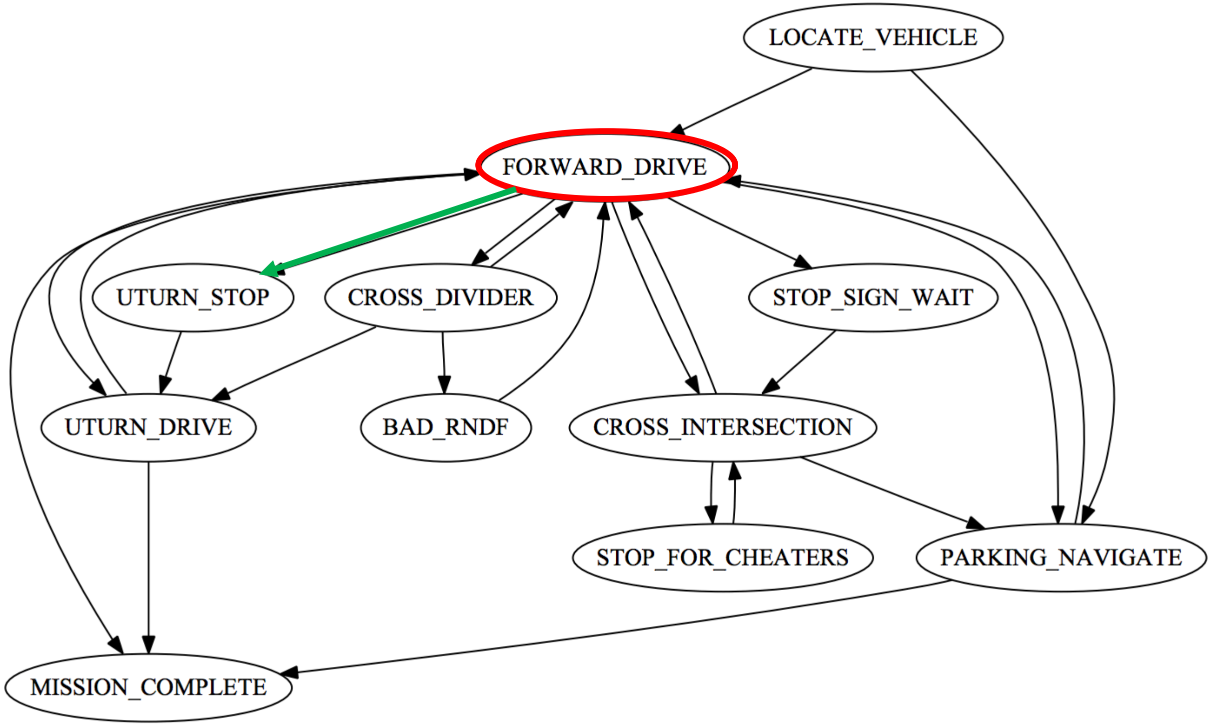
Mission Estimator State Machine

Finite State Machines (in a self driving car)



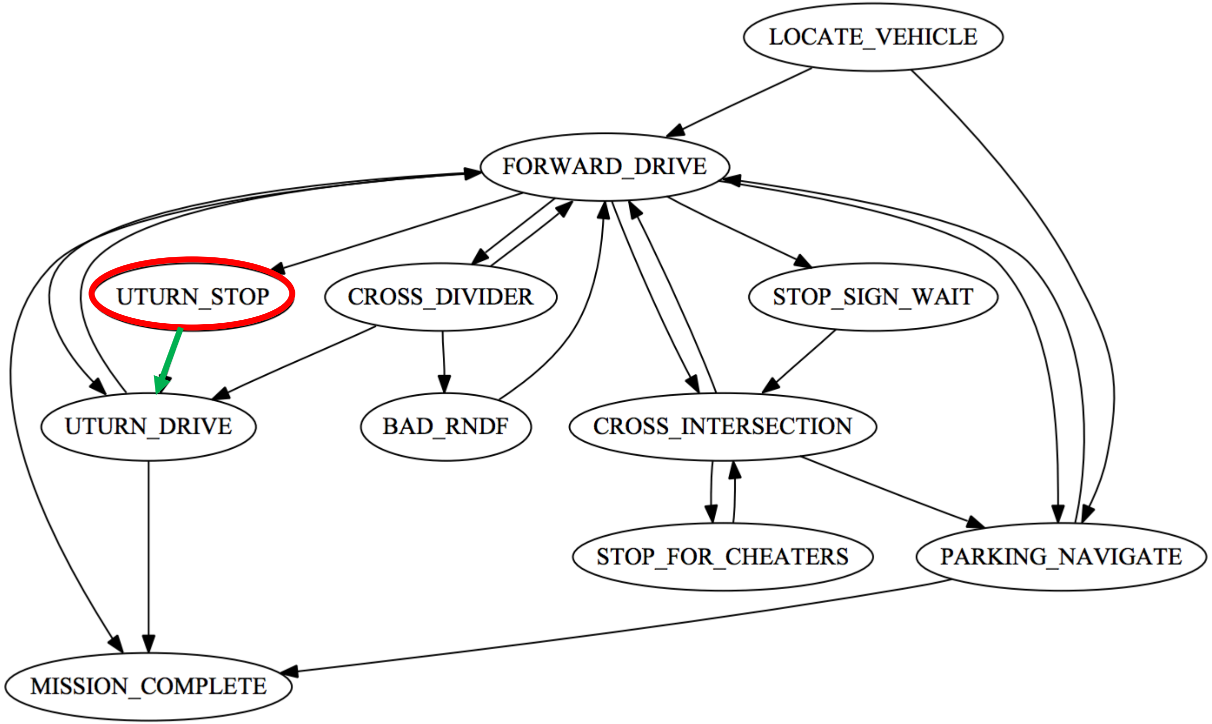
Stanford's DUC 2007 Behavior FSM ([link]())

Finite State Machines (in a self driving car)



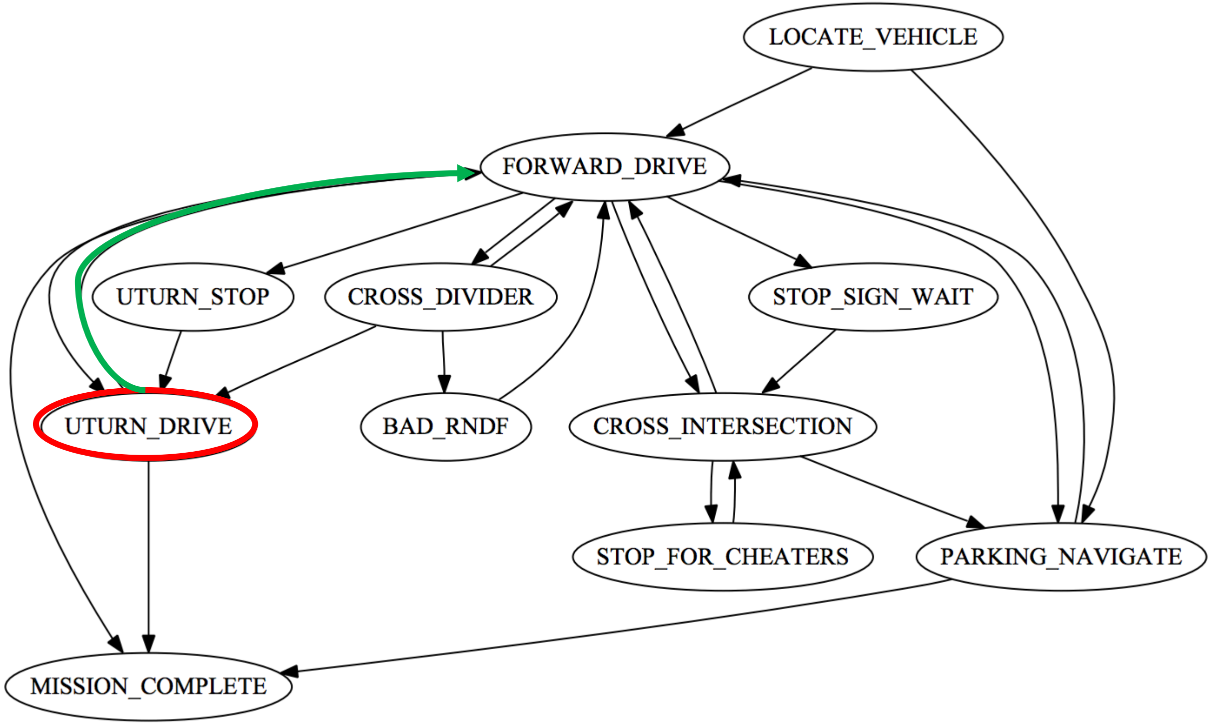
Stanford's DUC 2007 Behavior FSM (link)

Finite State Machines (in a self driving car)



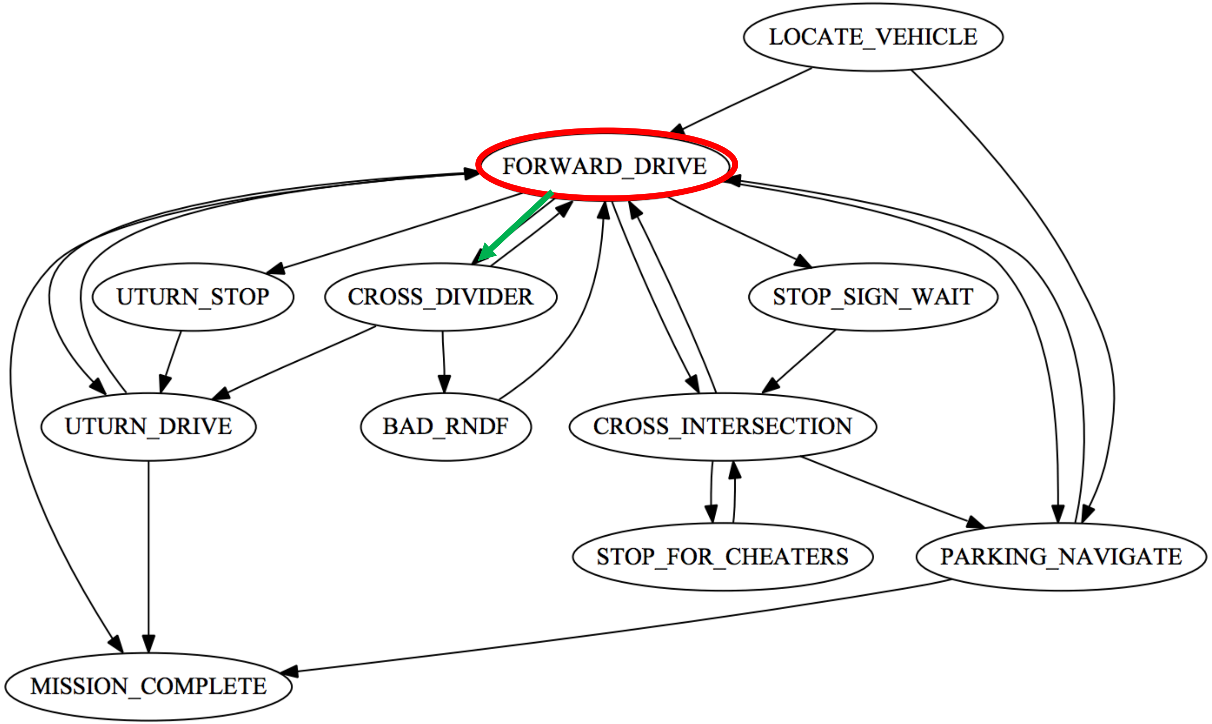
Stanford's DUC 2007 Behavior FSM (link)

Finite State Machines (in a self driving car)



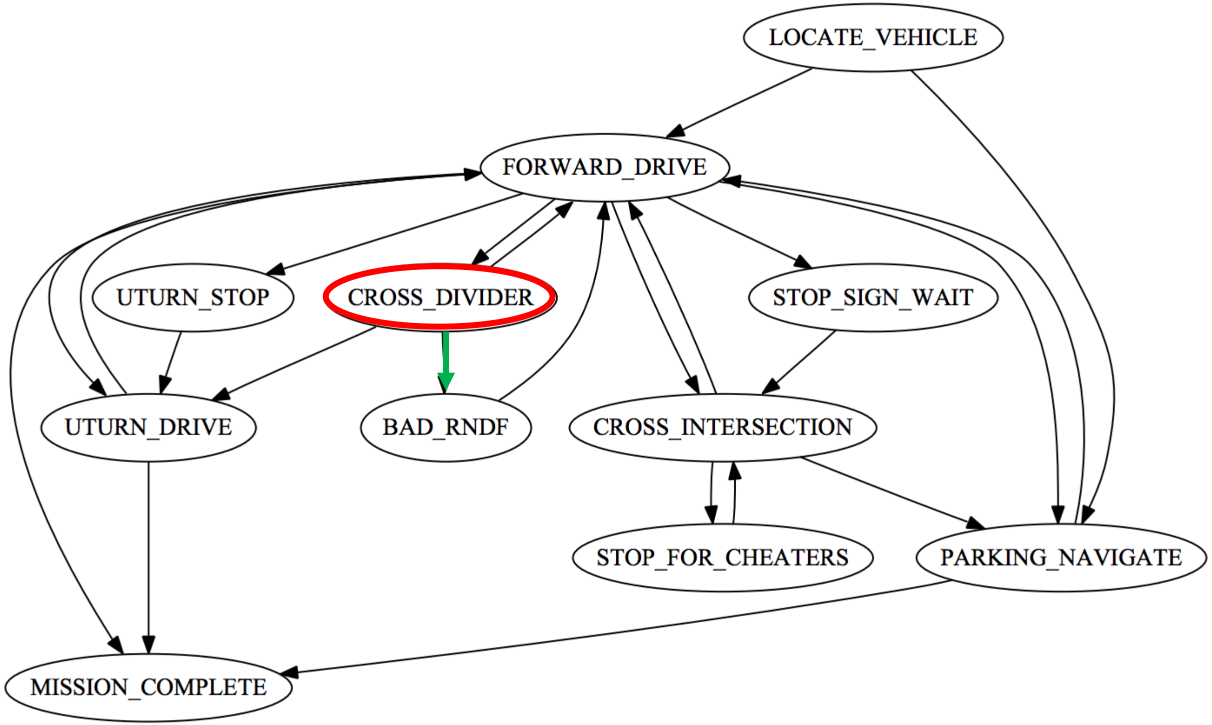
Stanford's DUC 2007 Behavior FSM (link)

Finite State Machines (in a self driving car)



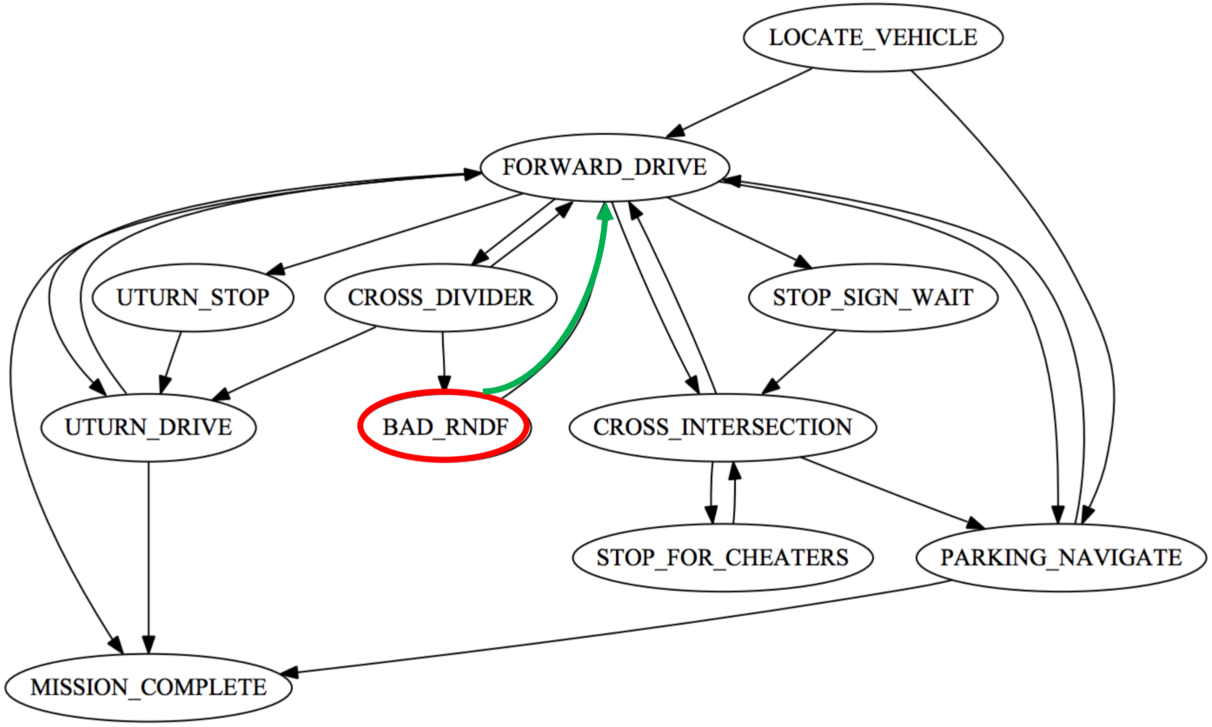
Stanford's DUC 2007 Behavior FSM ([link]())

Finite State Machines (in a self driving car)



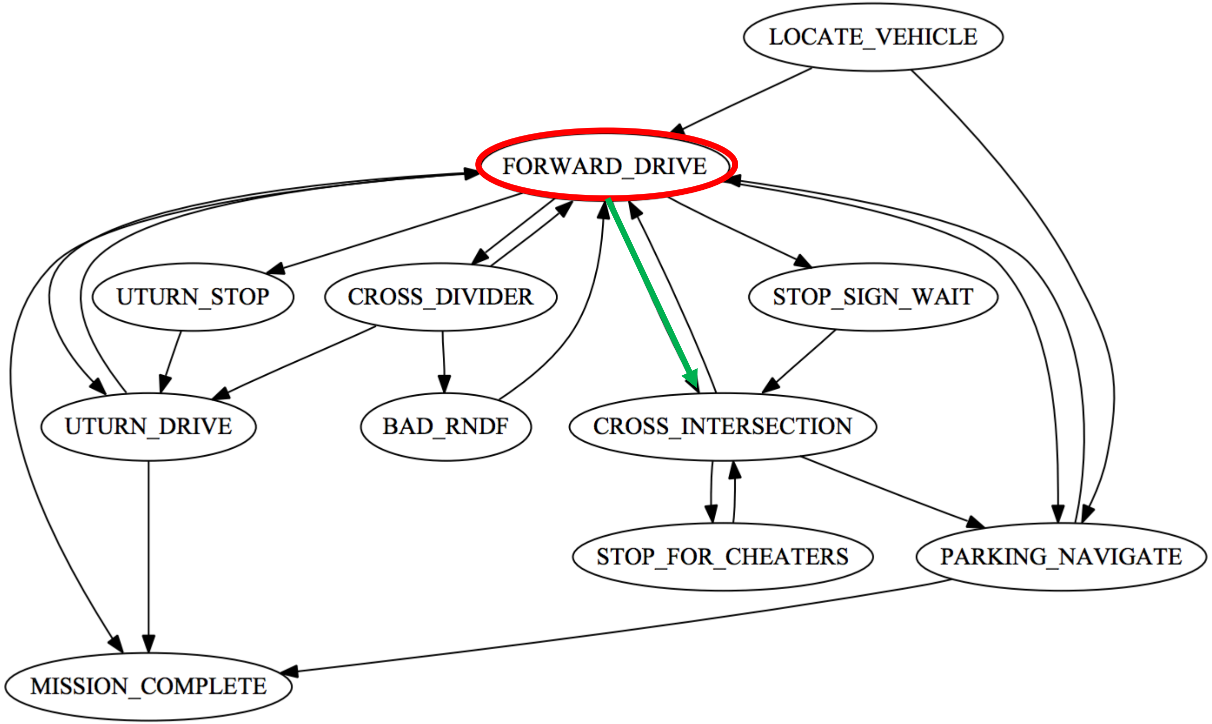
Stanford's DUC 2007 Behavior FSM (link)

Finite State Machines (in a self driving car)



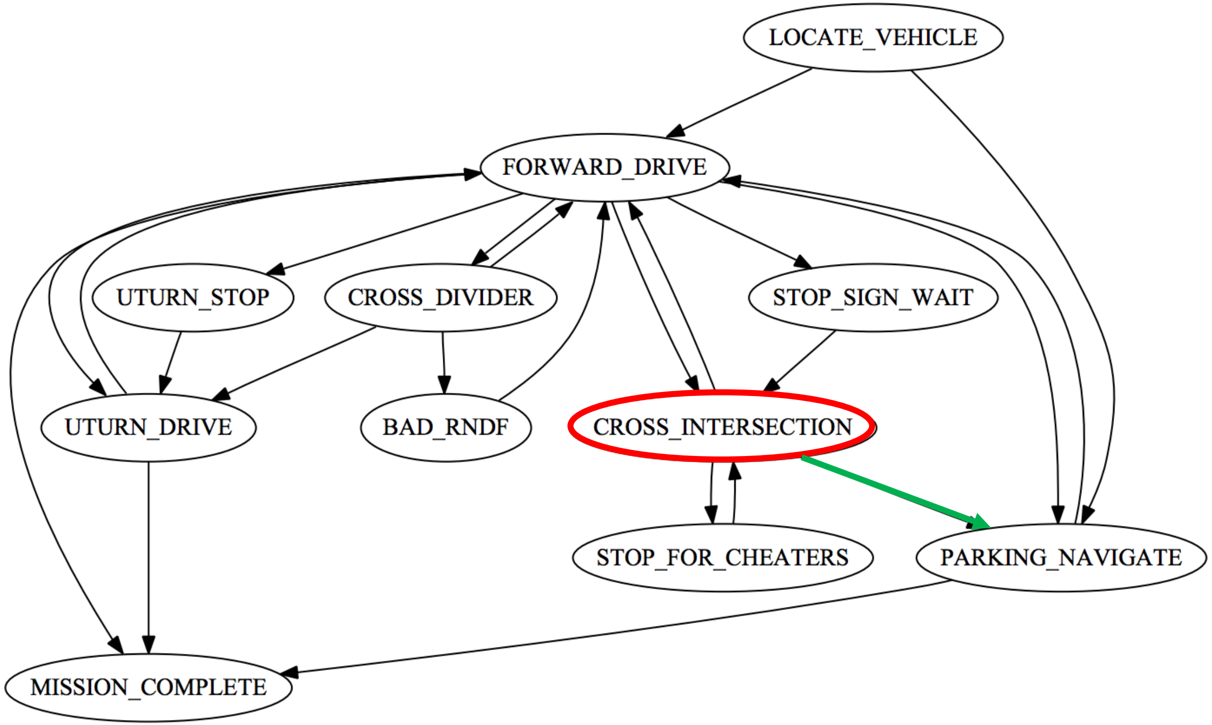
Stanford's DUC 2007 Behavior FSM (link)

Finite State Machines (in a self driving car)



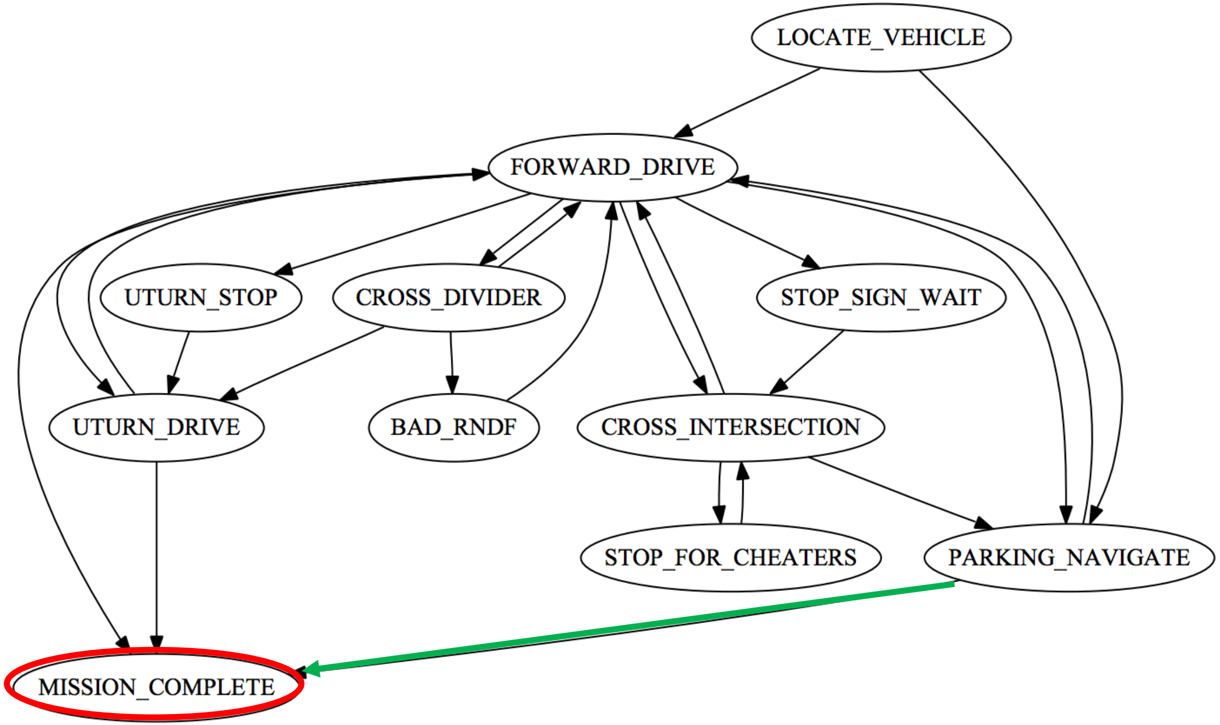
Stanford's DUC 2007 Behavior FSM (link)

Finite State Machines (in a self driving car)



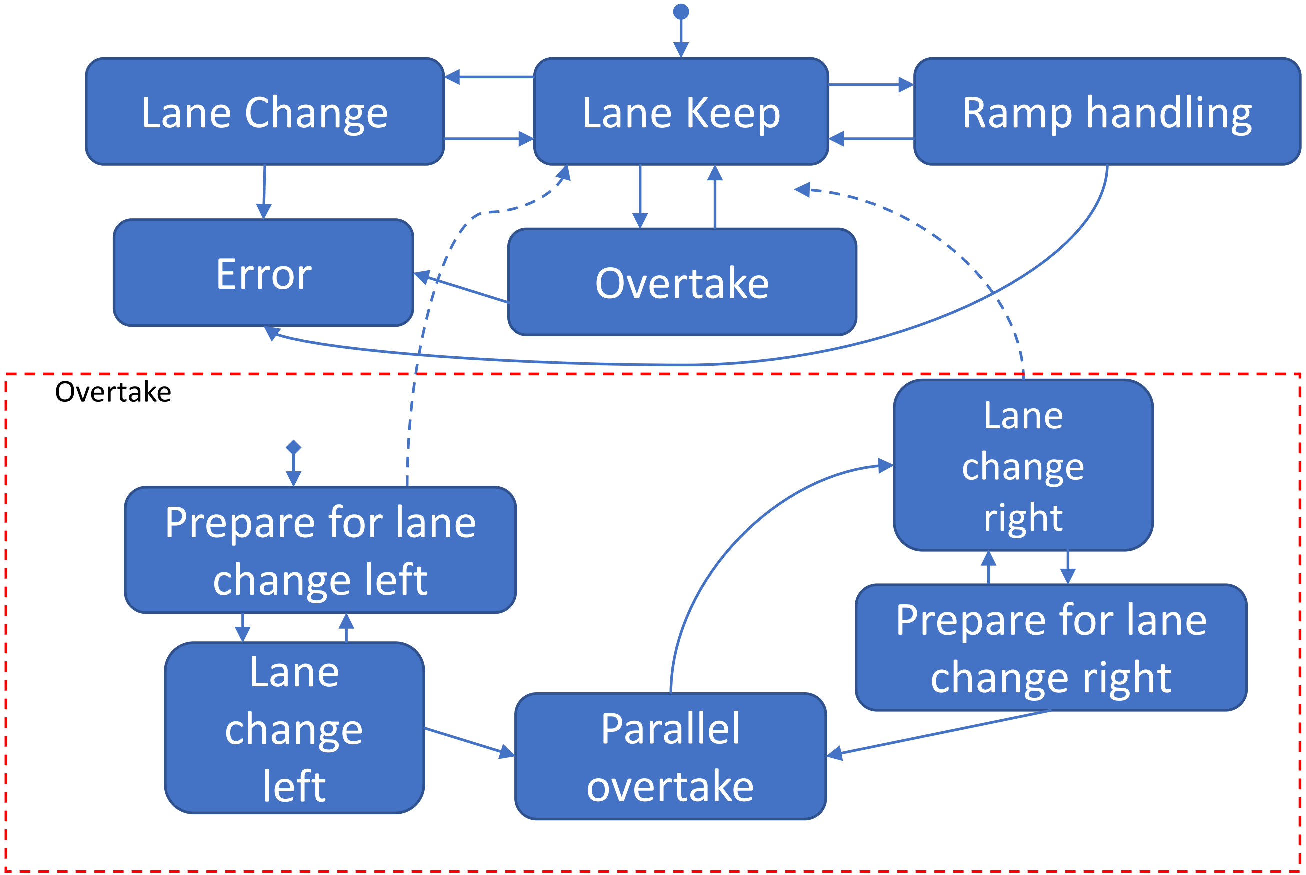
Stanford's DUC 2007 Behavior FSM (link)

Finite State Machines (in a self driving car)



Stanford's DUC 2007 Behavior FSM (link)

Finite State Machines (in a self driving car)



Finite State Machine (FSM): Cons

- A finite-state machine, or FSM for short, is a model of computation based on a hypothetical machine made of one or more states. Only a single state can be active at the same time, so the machine must transition from one state to another in order to perform different actions.

Limitation:

- Scalability: Can’t handle too complicated scenarios;
- Maintenance: easy make mistakes when there is a small change;
- Repeatability: almost impossible to reuse the same FSM in different application

Behavior Tree

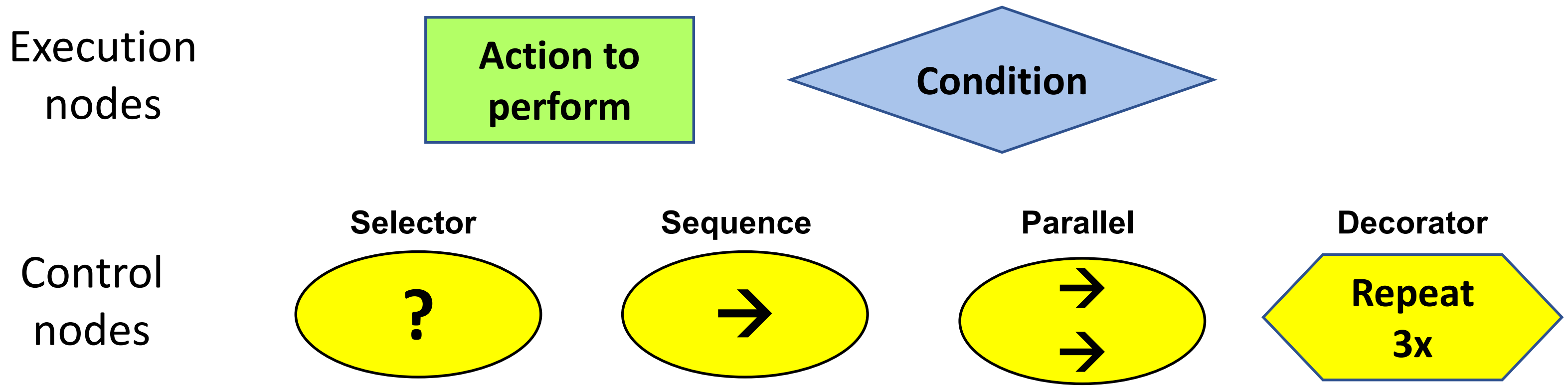
Behavior Trees (BT)

Behavior Trees are formulated as directed graphs with a tree structure and has the following characteristics:

- **Behavior Trees are trees**: They start at a root node and are designed to be traversed in a specific order until a terminal state is reached (success or failure).
- **Leaf nodes are executable behaviors**: Each leaf will do something, whether it's a simple check or a complex action, and will output a status (success, failure, or running). In other words, leaf nodes are where you connect a BT to the lower-level code for your specific application.
- **Internal nodes control tree traversal**: The internal (non-leaf) nodes of the tree will accept the resulting status of their children and apply their own rules to dictate which node should be expanded next.

Behaviour Tree: Terminology

- There are 6 basic types of nodes that make up behaviour trees and they are represented graphically:
- Behavior trees execute in discrete update steps known as ticks.
- After a node ticks, it returns a status to its parent, which can be Success, Failure, or Running.



Action node

- Action nodes are the leaves of the trees
- It performs a task and returns Success if the action is completed, Failure if the task could not be completed, and Running while the task is being performed.

Scenario: to overtake car in front while avoid oncoming cars

Actions needed to complete overtake:

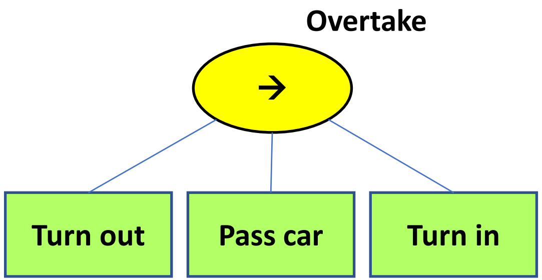
- Turn out — **Turn out**
- Pass car — **Pass car**
- Turn in — **Turn in**

Sequence node

- A sequence node ticks its children in sequence, trying to ensure that a number of sequential tasks are all performed.
- If any child return failure, the sequence has failed and it will propagate up.
- The sequence node only returns success if all children succeed.

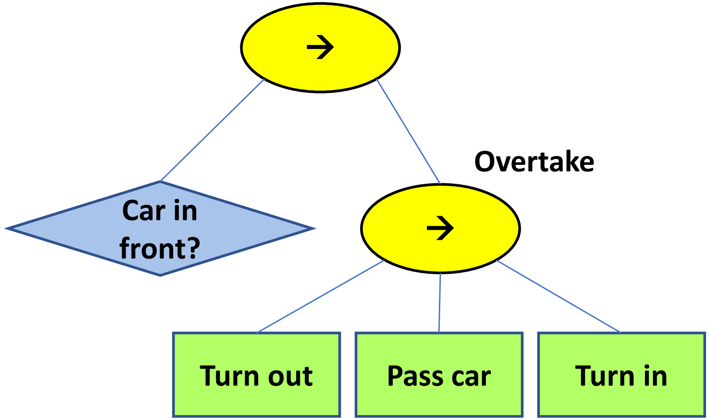
Algorithm 2 Sequence node

for each node $n \in$ *children* **do**
  $childstatus \leftarrow \text{tick}(n)$
  if $childstatus = \text{running}$ **then**
    return running
  else if $childstatus = \text{failure}$ **then**
    return failure
  end if
end for
return success



Condition node

- The Condition node is analogous to a simple if-statement.
- If the conditional check is true, the node returns Success and Failure if false.
- A Condition node will never return a Running status.

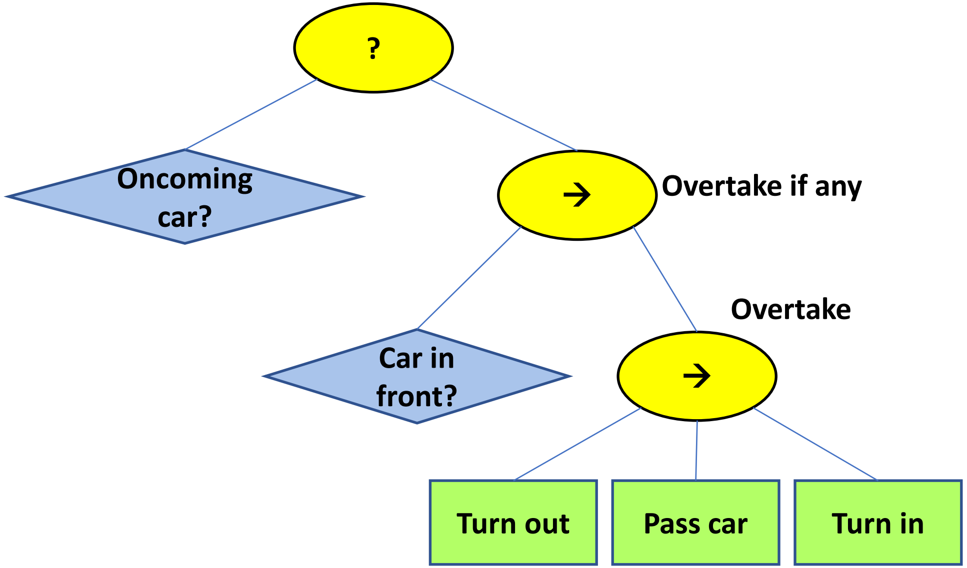


Selector node

- A selector node will begin to tick its children in order.
- If the first child fails, the execution continues to the following child and it is ticked.
- If a child succeeds, the selector also returns success and does not move on to the following children.

Algorithm 1 Selector node

```
for each node n ∈ children do
    childstatus ← tick(n)
    if childstatus = running then
        return running
    else if childstatus = success then
        return success
    end if
end for
return failure
```

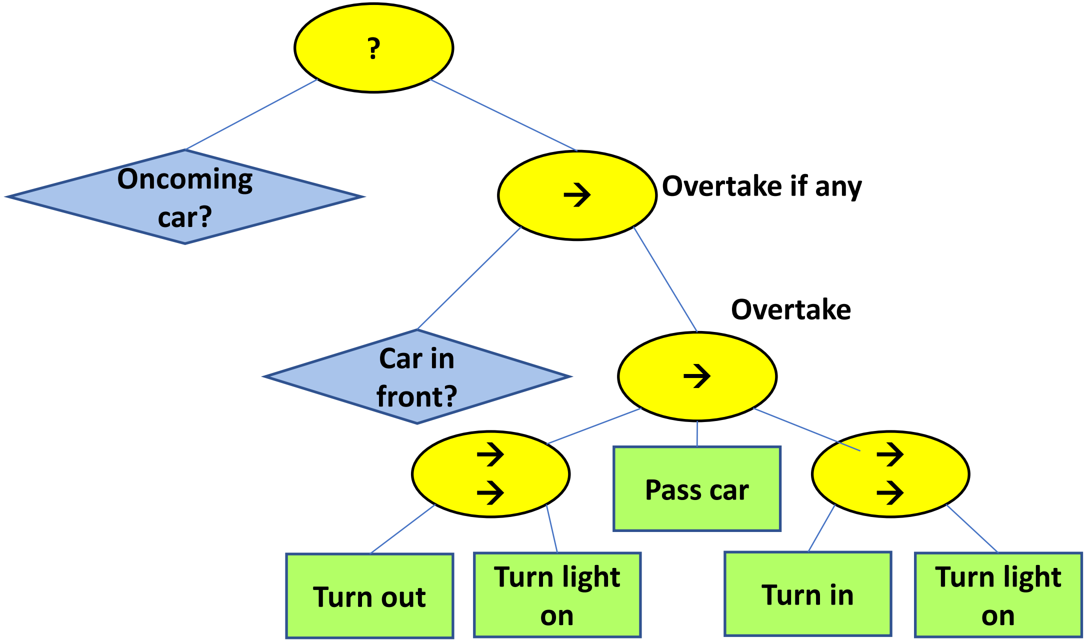


Parallel node

- A parallel node ticks all its children at the same time, allowing several Action nodes to enter a running state at the same time.
- The requirement for how many children need to succeed before the Parallel node itself reports success/failure can be customized on a per-instance basis.

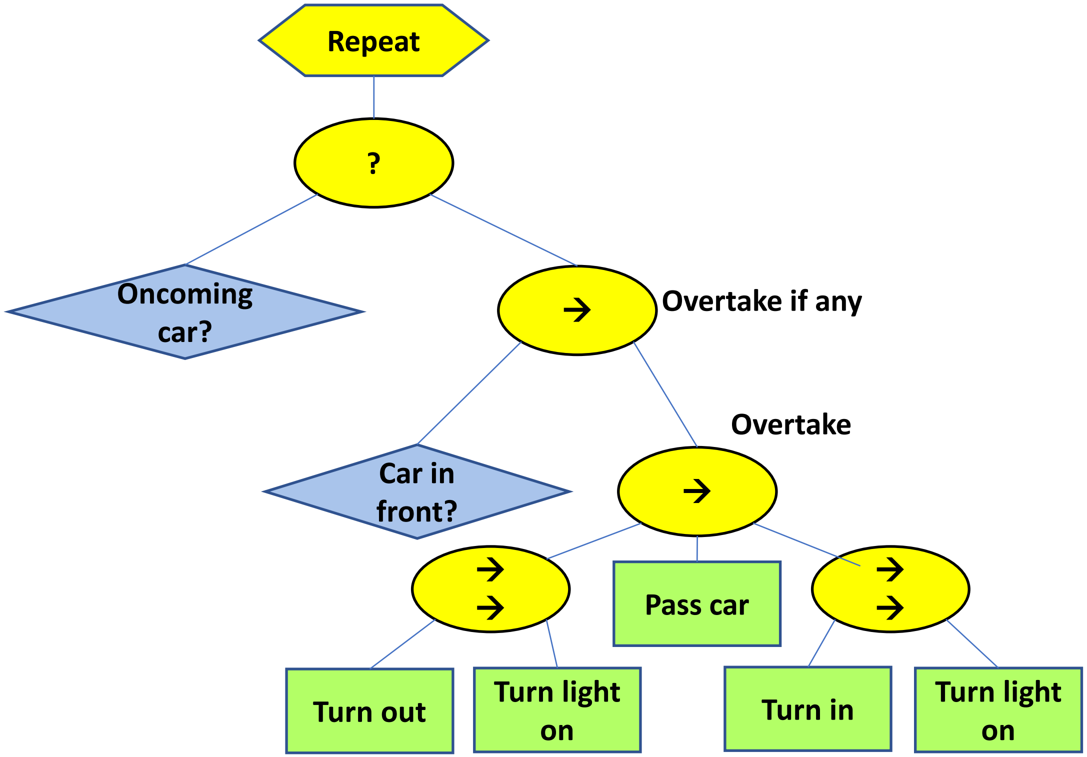
Algorithm 3 Parallel node

for each node $n \in$ *children* **do**
  $childstatus[i] \leftarrow \text{tick}(n)$
end for
if all_running($childstatus$) **then**
  return running
else if success_critera($childstatus$) **then**
  return success
else
  return failure
end if



Decorator node

- The decorator node wraps the functionality of the underlying child or subtree.
- It can for example influence the behavior of the underlying node(s) or modify the return state.
- E.g.
  - Inverter: flip the return status of its child from Success to Failure and vice versa.
  - Repeat: repeat the task multiple times



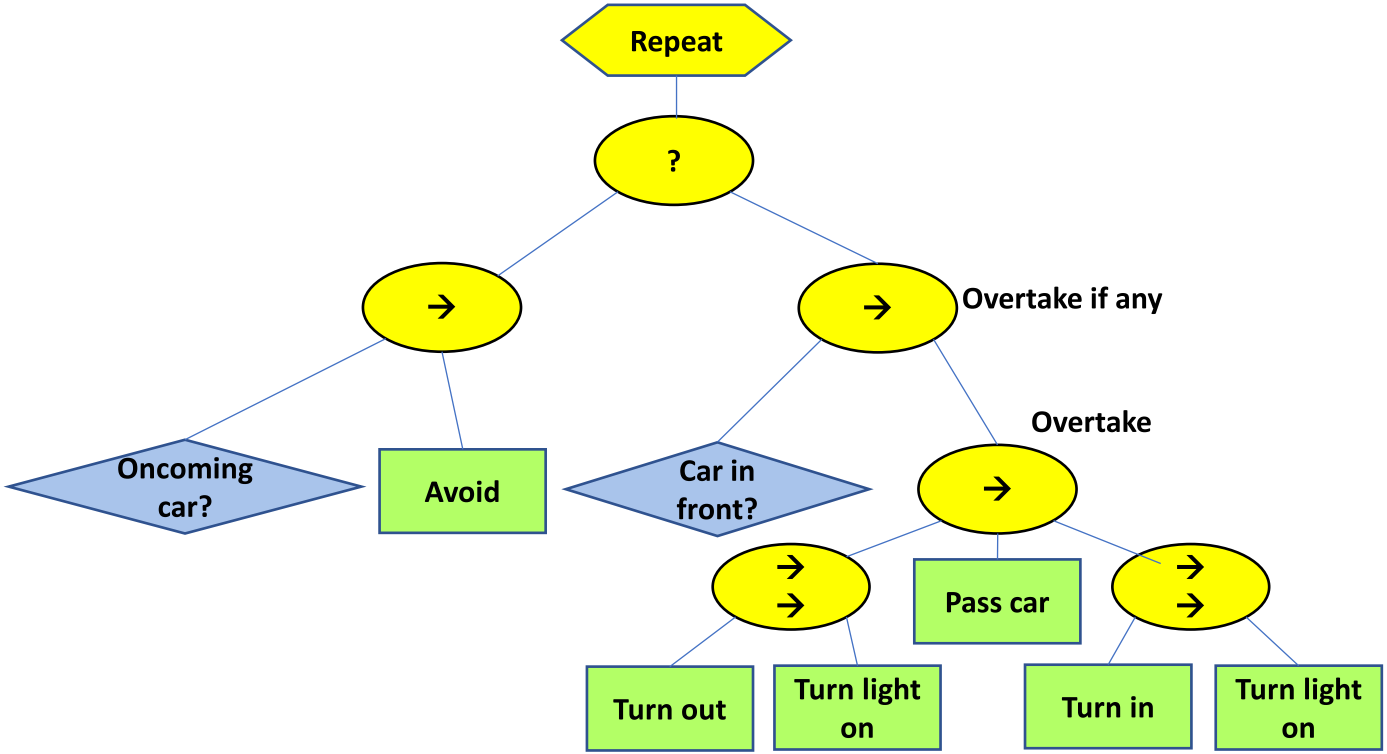
Algorithm 4 Decorator node

1: $childstatus \leftarrow \mathrm{tick}(n)$

2: **return** $\mathrm{func}(childstatus)$

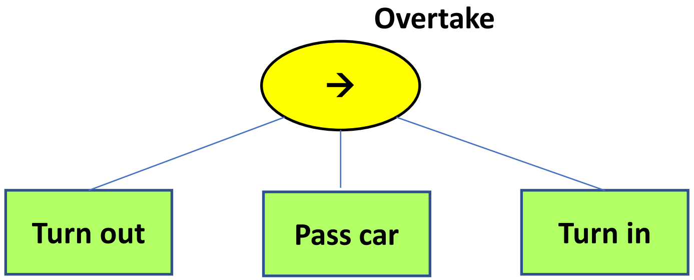
Example

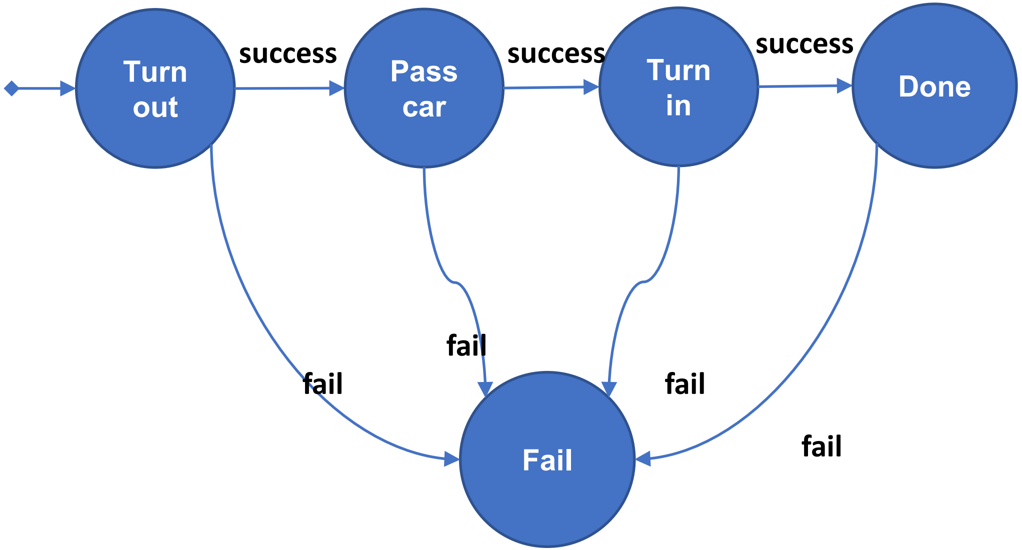
- Scenario: to overtake car in front while avoid oncoming cars.



Behavior Trees vs. Finite State Machine

- In theory, it is possible to express anything as a BT, FSM, one of the other abstractions, or as plain code. However, each model has its own advantages and disadvantages in their intent to aid design at larger scale.
- Specific to BTs vs. FSMs, there is a tradeoff between **modularity** and **reactivity**. Generally, BTs are easier to compose and modify while FSMs have their strength in designing reactive behaviors

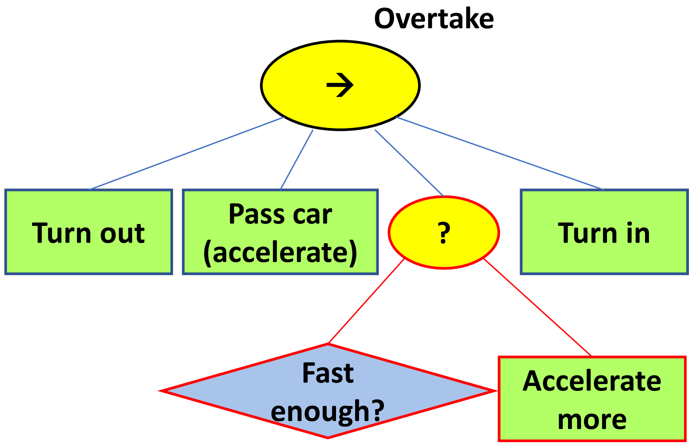


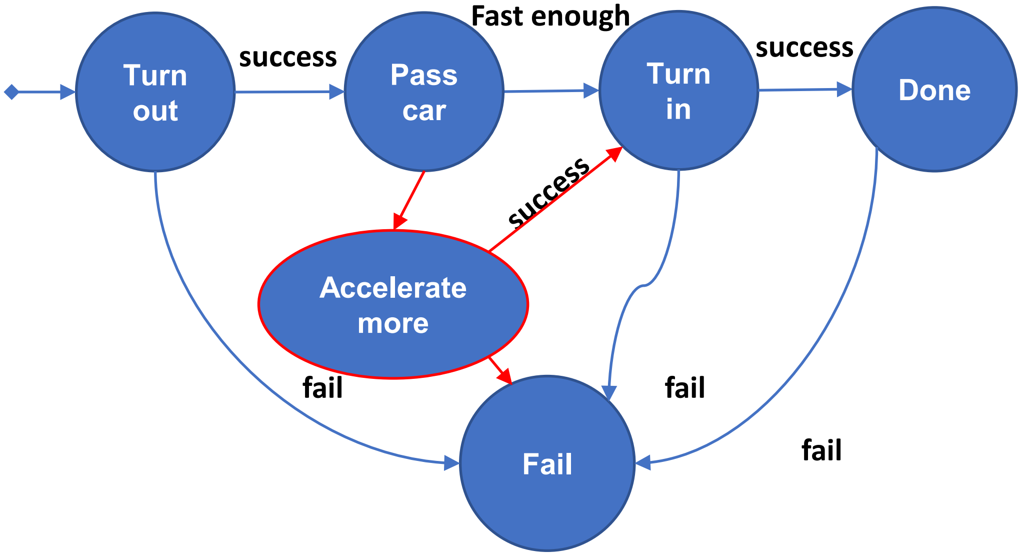


Behavior Trees vs. Finite State Machine

In theory, it is possible to express anything as a BT, FSM, one of the other abstractions, or as plain code. However, each model has its own advantages and disadvantages in their intent to aid design at larger scale.

- Specific to BTs vs. FSMs, there is a tradeoff between **modularity** and **reactivity**. Generally, BTs are easier to compose and modify while FSMs have their strength in designing reactive behaviors

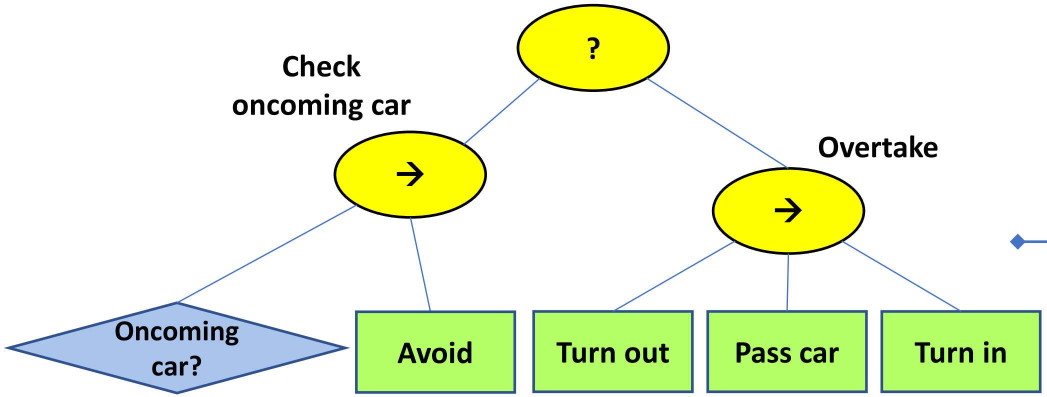


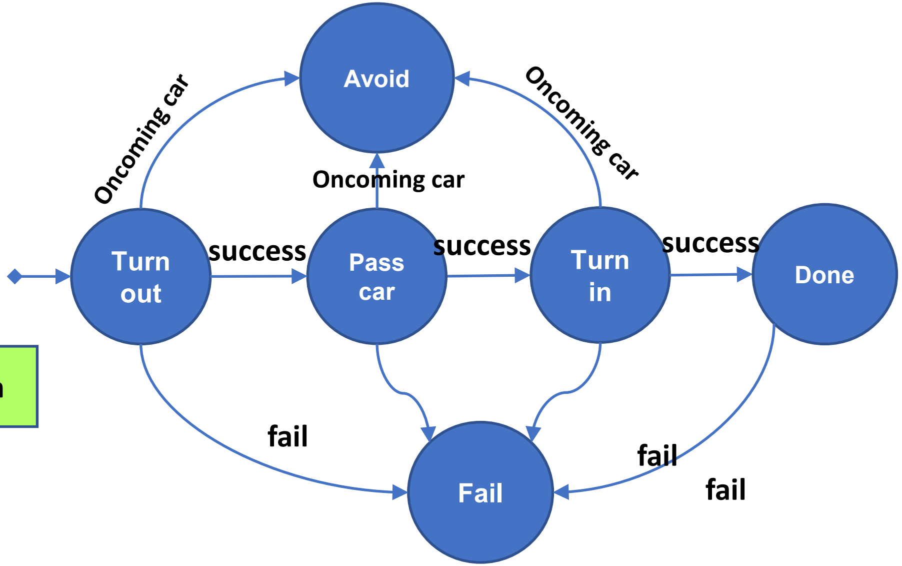


Behavior Trees vs. Finite State Machine

In theory, it is possible to express anything as a BT, FSM, one of the other abstractions, or as plain code. However, each model has its own advantages and disadvantages in their intent to aid design at larger scale.

- Specific to BTs vs. FSMs, there is a tradeoff between **modularity** and **reactivity**. Generally, BTs are easier to compose and modify while FSMs have their strength in designing reactive behaviors





POMDP