LECTURE 3 - 2

Designing a Reactive Implementation

Introduction to AI Robotics (Ch. 5)

Quote of the Week

"A common mistake people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools."

ANNOUNCEMENTS

- Lab 3 - Wall Following (PD Control) is due on *Thursday, 3/25/10*
- The lab memo and code is due on Angel by midnight on *Thursday, 3/25/10*
- Quiz 6 on Ch. 5, Lecture 3-2 on *Monday, 3/29/10*

OBJECTIVES

Upon completion of this lecture the student should be able to:

- Use schema theory to design and program behaviors
- Describe a complete behavioral system
- Draw a behavior table
- Define the terms: *releaser, perceptual schema, motor schema for a behavior*
- Describe the two methods for assembling and coordinating primitive behaviors
- Be able to represent a sequence of behaviors using a state diagram

EMERGENT BEHAVIOR

- The reactive movement involves a robot running a very small set of behaviors which were combined internally to produce an overall *emergent behavior*
- All robot behaviors result from the interaction of the robot's controller and the environment.
- If these components are simple then the resulting behavior may be predictable
- If the environment is dynamic or the controller has several interacting components then the resultant robot behavior may be a surprise
- If the unexpected behavior has some structure, pattern, or meaning it is often called *emergent*

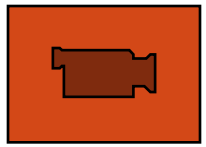
WALL FOLLOWING EMERGENT BEHAVIOR

- Rules:
  - If left whisker bent, turn right
  - If right whisker bent, turn left
  - If both whiskers bent, back up and turn left
  - Otherwise, keep going
- Based upon these behaviors the robot will follow a wall although the robot knows nothing about walls and it is not explicit in the rules

THE WHOLE IS GREATER THAN THE SUM OF ITS PARTS

- Emergent behavior appears to produce more than what the robot was programmed to do.
- We get more than we built in or something for nothing
- Roboticists sometimes exploit this potential to design clever and elegant controllers
- In particular, reactive and behavior-based systems are designed to take advantage of such interactions

FLOCKING EXAMPLE



- To make robots flock together when they cannot communicate with each other, have their own rules, and own local sensory data, use emergent behavior
- This can be achieved with the following rules
  - Don’t get too close to other robots or obstacles
  - Don’t get too far from other robots
  - Keep moving if you can
- To get the robots to move to a certain destination use another rule which moves each robot toward the goal point

COMPONENTS OF EMERGENCE

- Emergence depends on two components
  - Existence of an external observer to see the behavior and describe it
  - Access to the innards of the controller to verify that the behavior is not explicitly specified in the system
- Alternate definition of emergent behavior
  - *A structured (patterned, meaningful) behavior that is apparent at one level of the system (observer's viewpoint) but not at another (controller's/robot's viewpoint)*

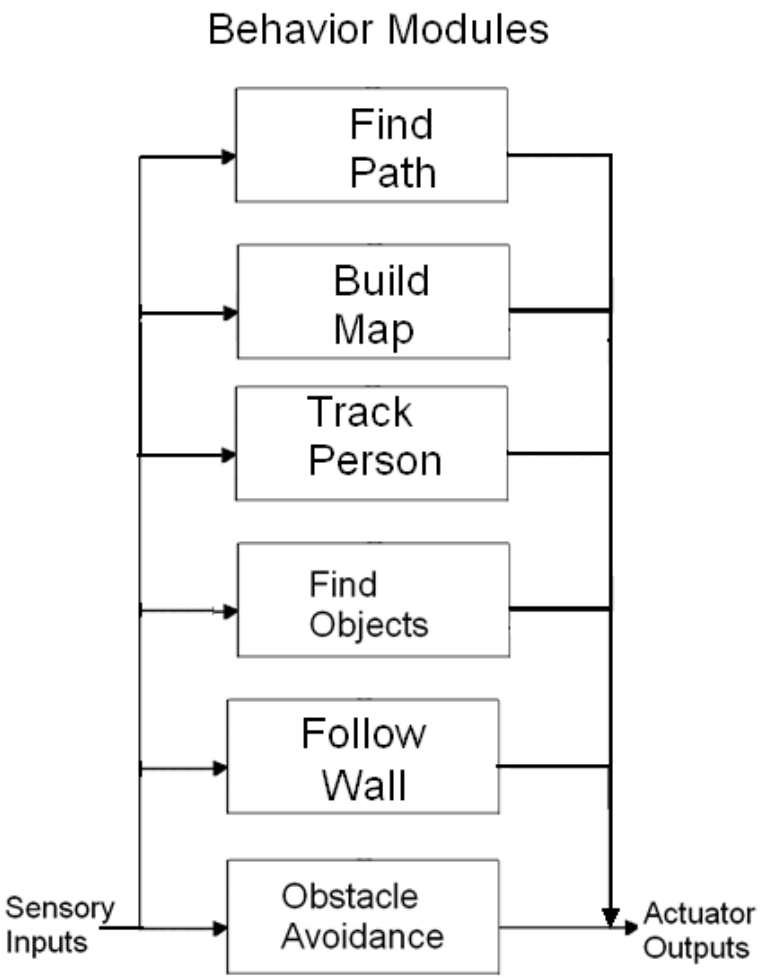
ARCHITECTURES AND EMERGENCE

- Different control architectures affect the likelihood of generating emergent behavior in different ways
- *Reactive* and *behavior-based systems* employ parallel rules and behaviors which interact with each other and the environment which provide the perfect foundation for exploiting emergent behavior
- Deliberative and hybrid systems do not have parallel interaction and would require environment structure and thus minimize emergence

BEHAVIOR-BASED CONTROL

- *Behavior-based control (BBC)* incorporates the best of reactive systems but does not involve a hybrid solution
- BBC is inspired by biological systems
- BBC was inspired by the fact that
  - Reactive systems are too inflexible, incapable of representation, adaptation, or learning
  - Deliberative systems are too slow and cumbersome
  - Hybrid systems require complex means of interaction among the components
- BBC is closest to reactive control and farthest from deliberative control

BEHAVIOR-BASED ARCHITECTURE



BEHAVIOR DEFINITION

- *Behaviors* achieve and /or maintain particular goals (i.e. homing, wall following)
- *Behaviors* are time extended, not instantaneous. They may take some time to achieve and/or maintain their goals
- *Behaviors* can take input from sensors and also other behaviors. Behaviors can send outputs to effectors and to other behaviors
- *Behaviors* are more complex than actions. A reactive system may use simple actions like *stop*, *turn right*.
- *BBC* may have behaviors such as *find object, follow wall, get recharged, hide from the light*.

ABSTRACTION

- Behaviors can be designed at a variety of levels of detail or description. This is called *their level of abstraction.*
- To be abstract is to take details away and make things less specific
- Behaviors can take different amounts of time and computation
- Behaviors are flexible and this is one of the key advantages of BBC
- The power and benefit is also in the way they are organized and put together in the control system

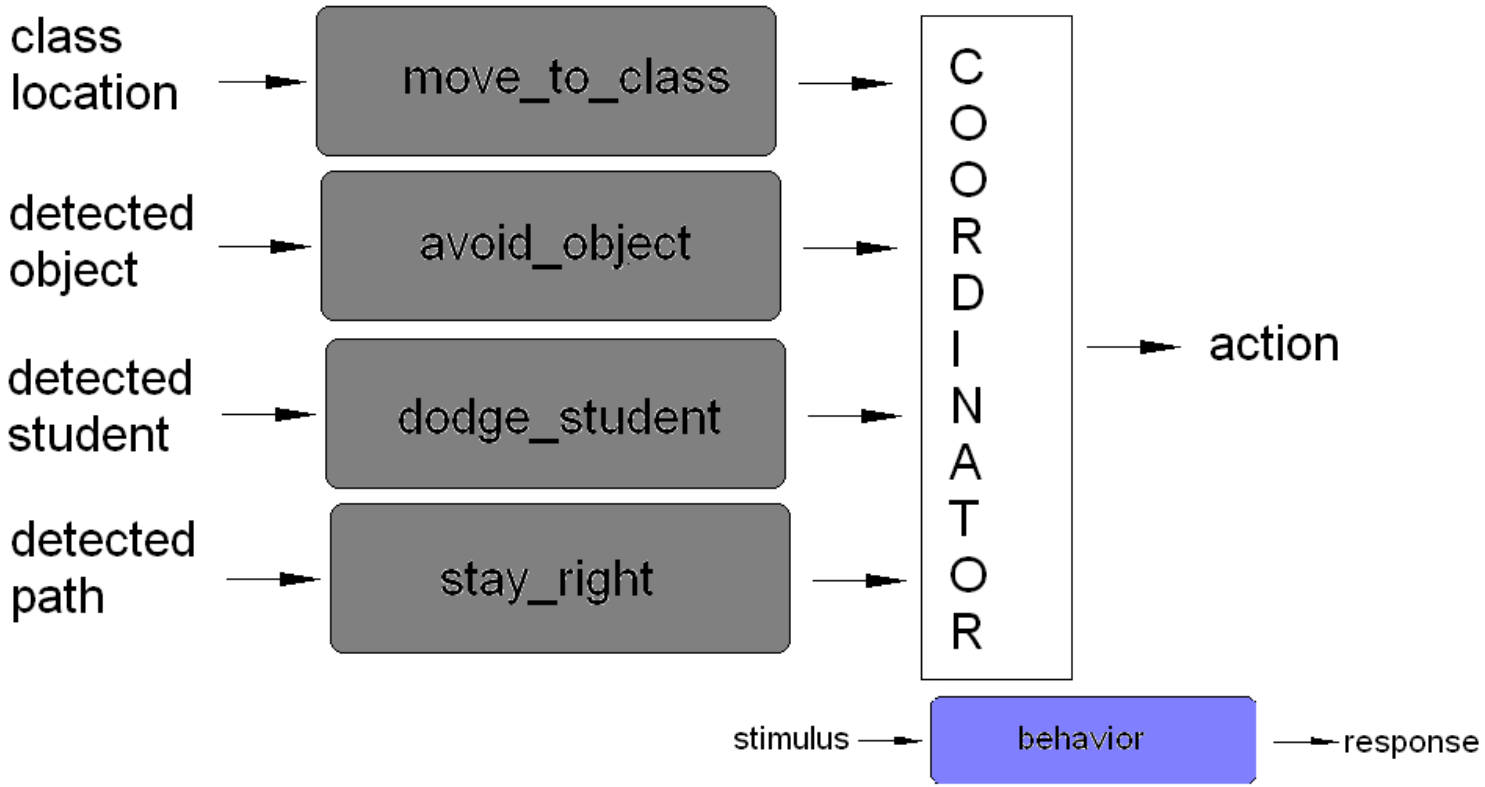
PRIMITIVE AND ABSTRACT BEHAVIORS

- A *primitive behavior* is composed of only one perceptual schema and one motor schema so there is no need for a *coordinated control program*
- Primitive behaviors are *monolithic* that only do one thing and are programmed as a single method
- Behaviors assembled from other behaviors or that have multiple perceptual schema and motor schema are *abstract behaviors*
- *A behavior puts the percept by the perceptual schema in a local place where the motor schema can get it*

EXPRESSING ROBOTIC BEHAVIOR

- There are several methods available for expressing robotic behavior including:
  - Stimulus-response (SR) diagrams
    - Use for graphic representation of specific behavioral configurations
  - Functional Notation
    - Used for clarity in design of systems
  - Finite state acceptor (FSA) diagrams
    - Used for temporal sequencing of behaviors

SR DIAGRAM FOR CLASSROOM NAVIGATION ROBOT



FUNCTIONAL NOTATION FOR CLASSROOM NAVIGATION ROBOT

B(S) = R

The behavior b when given stimulus s yields response r

Coordinate-behaviors

[

move_to_class(detect-class-location),

avoid_objects(detect-objects),

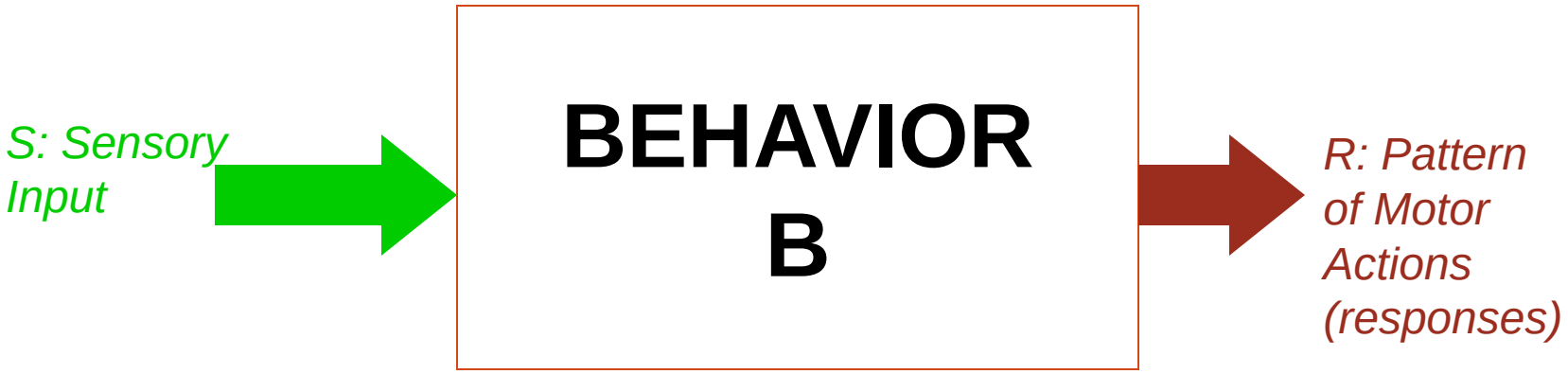
dodge_students (detect-students),

stay_to_right(detect-path)

return motor_response

]

BEHAVIOR DEFINITION

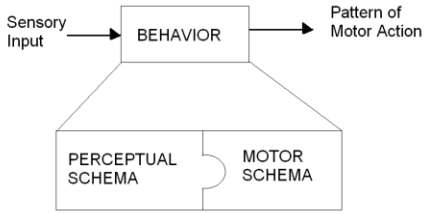


A behavior is a mapping of sensory inputs to a pattern of motor actions which are then used to achieve a task.

Notation: **B(S)=R**

BEHAVIORS AS OBJECTS IN OBJECT-ORIENTED PROGRAMMING

- *Objects* consist of data and methods
- A *schema* consists both of the knowledge of how to act and/or perceive
- A behavior is a schema that consists of a *perceptual schema* and *motor schema*
- A *schema* will be a *class*
- The schema class has an optional method called a *coordinated control program*
- The *coordinated control program* is a function that coordinates any methods or schemas in the derived class



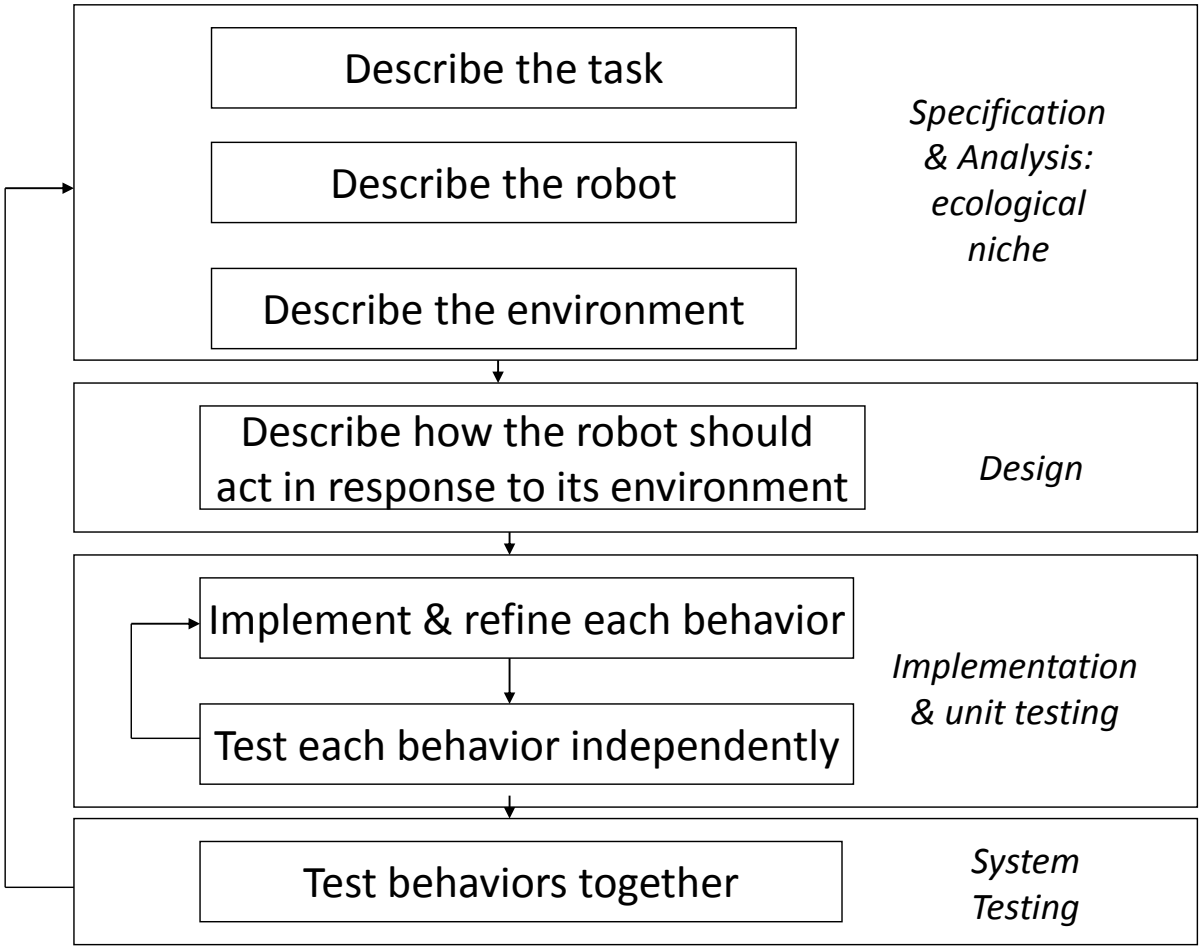
BBC DESIGN PRINCIPLES

- Behaviors are typically executed in parallel/concurrently
- Networks or behaviors are use to store state and construct world models/representations
- Behaviors are designed to operate on compatible time scales
- BBC have the following key properties
  - The ability to react in real time
  - The ability to use representations to generate efficient behavior
  - The ability to use uniform structure and representation through the system

TIME AND MODULARITY AND REPRESENTATION

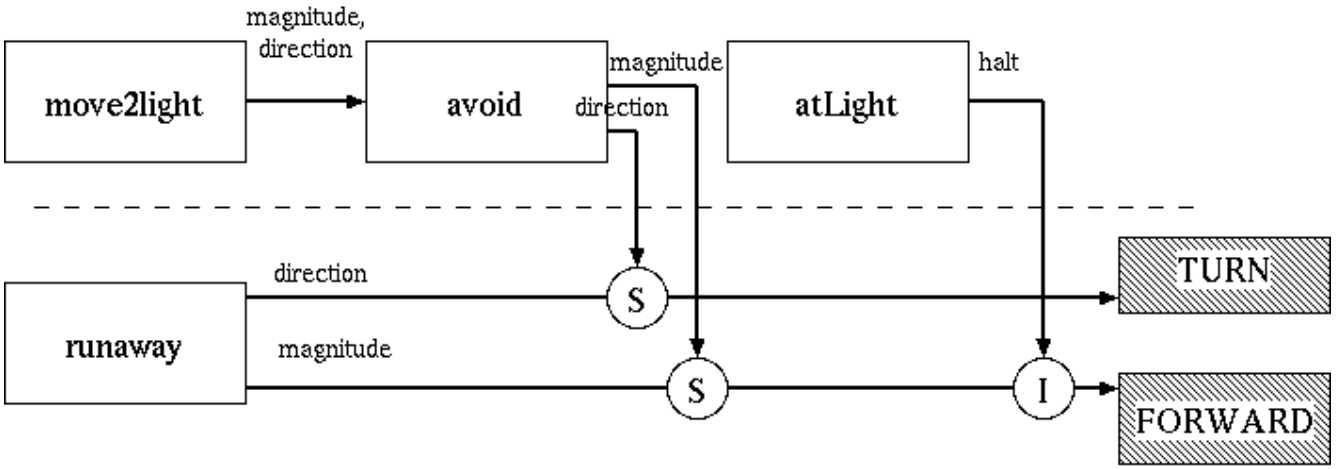
- BBC is modular by using a collection of behaviors
- These behaviors are relatively similar in terms of execution time
- It is based on reactive philosophy where behaviors are incrementally added to the system
- The behaviors are executed concurrently in parallel
- Behaviors are activated in respond to internal and/or external conditions
- Behaviors are more expressive than simple reactive rules
- Behaviors are more complex and more flexible than reactive rules and can be used in clever ways to program robots

DESIGNING A BEHAVIORAL SYSTEM



PHOTOPHILIC BEHAVIOR DESIGN

Design a robot that is attracted to light. It will head toward the light and if it encounters an obstacle, turn left or right in the direction to favor the light. If there is not light the robot will sit and wait. If an obstacle appears the robot will turn and run away



notice: unlike examples in book, robot doesn't have to go a fixed speed

PHOTOPHILIC: BEHAVIOR TABLE

Design a robot that is attracted to light. It will head toward the light and if it encounters an obstacle, turn left or right in the direction to favor the light. If there is not light the robot will sit and wait. If an obstacle appears the robot will turn and run away

Releaser	Behavior	Motor Schema	Percept	Perceptual Schema
Light	Photophilic	move2Light()	light: direction & strength	brightness(dir), atLight()
Range	Obstacle avoidance	avoid() runaway()	proximity	obstacle()

FINITE STATE AUTOMATA (FSA)

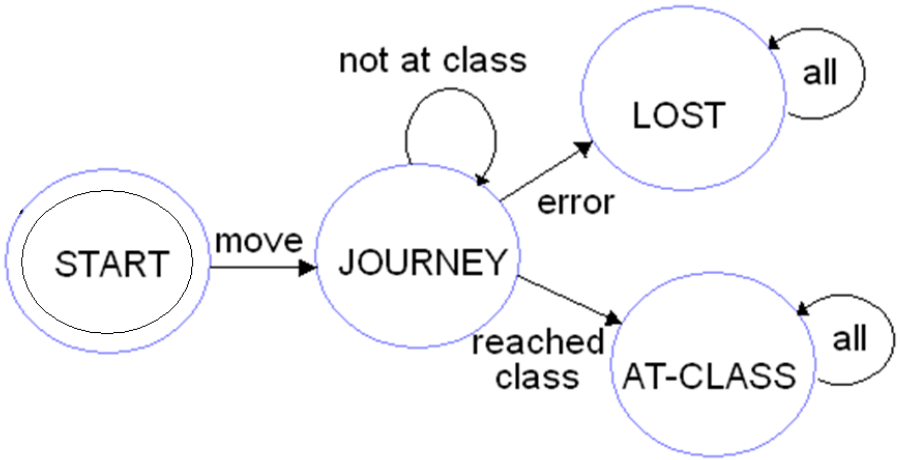
- *FSA* are a set of mechanisms that describe what a program should be doing a given time or circumstance
- The FSA can be written as a *table* or *state diagram*

State, q	σ, input	δ (q, σ)
journey	not at class	journey
journey	Error	Lost
journey	reached class	at class
at class	all	at class
lost	all	Lost

FSA	Behavioral analog
M	Finite State Machine
K	All the behaviors for a task
Σ	The releasers for each behavior in K
δ	The function that computes the new state, transition function
s	The behavior the robot starts in when it is turned on
F	The behaviors the robot must reach to terminate
q	Current state
σ	Current input

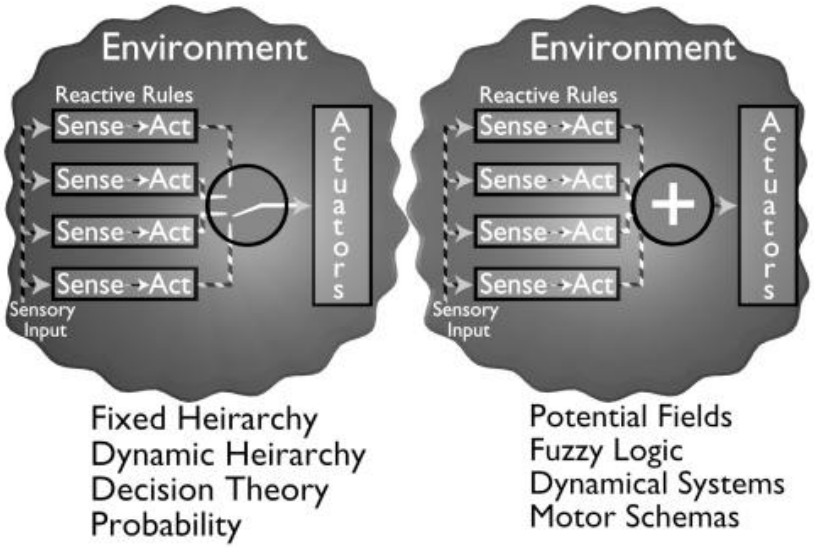
FINITE STATE ACCEPTOR (FSA) DIAGRAMS FOR CLASSROOM NAVIGATION ROBOT

- FSA diagrams describe the aggregations and sequences of behaviors
- It explicitly shows the active behavior at any given time and the transitions between them
- the circle denotes the state where behavior is active and the arrows represent the stimulus input and response
- Note that Journey includes move_to_class, avoid_objects, dodge_students and stay_right



BEHAVIOR ARBITRATION VERSUS BEHAVIOR FUSION

- *Behavior Arbitration* is picking one behavior /action
- *Behavior Fusion* is combining multiple behaviors /actions



BEHAVIOR ARBITRATION

- Arbitration-based behavior coordination is also called *competitive behavior* because candidate behaviors compete but only one can win
- In a *fixed priority hierarchy*, the behaviors have pre-assigned priorities
- In a *dynamic hierarchy*, the behavior priorities change at run-time
- *Subsumption architecture* uses a fixed priority hierarchy of behaviors. Some hybrid systems also employ fixed priority hierarchy

BEHAVIOR FUSION

- *Behavior fusion* is the process of combining multiple possible candidates into a single output action/behavior
- *Behavior fusion* is also called a cooperative method because it combines outputs of multiple behaviors to produce a final result
- This result may be an existing behavior or a new one (*emergent behavior*)
- There could be weighting or have some logic in the system to prevent certain combinations and outcomes