INTELLIGENT AGENTS AND SEARCHS

Unit 1

Dr. Ashaq Hussain Bhat

University Institute of Computing

Chandigarh University

OUTLINE

- AGENTS AND ENVIRONMENTS
- RATIONALITY
- PEAS (PERFORMANCE MEASURE, ENVIRONMENT, ACTUATORS, SENSORS)
- ENVIRONMENT TYPES
- AGENT TYPES

AGENTS

- AN AGENT IS ANYTHING THAT CAN BE VIEWED AS PERCEIVING ITS ENVIRONMENT THROUGH SENSORS AND ACTING UPON THAT ENVIRONMENT THROUGH ACTUATORS

- HUMAN AGENT:
  - EYES, EARS, AND OTHER ORGANS FOR SENSORS;
  - HANDS, LEGS, MOUTH, AND OTHER BODY PARTS FOR ACTUATORS

- ROBOTIC AGENT:
  - CAMERAS AND INFRARED RANGE FINDERS FOR SENSORS
  - VARIOUS MOTORS FOR ACTUATORS

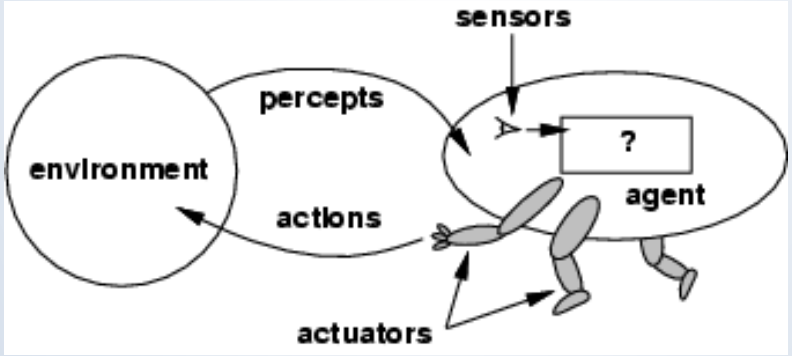
Agent Architecture

An agent can be represented by the following simple structure:

- **Percept**: The agent perceives the environment using its sensors.
- **Action**: Based on what it perceives, the agent chooses an action.
- **Environment**: The action changes the environment, which may affect the agent's future perceptions.

This cycle of perception-action is fundamental to how agents operate, and it can be repeated continuously in a loop. This interaction forms the basis of intelligent decision-making in AI agents.

AGENTS AND ENVIRONMENTS

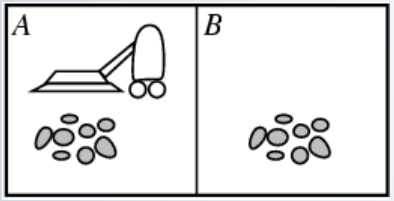


- THE AGENT FUNCTION MAPS FROM PERCEPT HISTORIES TO ACTIONS:

$$[F: \mathcal{P}^{\star} \rightarrow \mathcal{A}]$$

- THE AGENT PROGRAM RUNS ON THE PHYSICAL ARCHITECTURE TO PRODUCE *F*
- AGENT = ARCHITECTURE + PROGRAM

VACUUM-CLEANER WORLD



Demo: http://www.ai.sri.com/~oreilly/aima3ejava/aima3ejavademos.html

- PERCEPTS: LOCATION AND CONTENTS, E.G., [A,DIRTY]
- ACTIONS: *LEFT*, *RIGHT*, *SUCK*, *NOOP*
- *AGENT'S FUNCTION → LOOK-UP TABLE*

Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
⋮	⋮

RATIONAL AGENTS

- RATIONALITY
  - PERFORMANCE MEASURING SUCCESS
  - AGENTS PRIOR KNOWLEDGE OF ENVIRONMENT
  - ACTIONS THAT AGENT CAN PERFORM
  - AGENT'S PERCEPT SEQUENCE TO DATE

- RATIONAL AGENT: FOR EACH POSSIBLE PERCEPT SEQUENCE, A RATIONAL AGENT SHOULD SELECT AN ACTION THAT IS EXPECTED TO MAXIMIZE ITS PERFORMANCE MEASURE, GIVEN THE EVIDENCE PROVIDED BY THE PERCEPT SEQUENCE AND WHATEVER BUILT-IN KNOWLEDGE THE AGENT HAS.

EXAMPLES OF RATIONAL CHOICE

- SEE FILE: INTRO-CHOICE.DOC

RATIONALITY

- RATIONAL IS DIFFERENT FROM OMNISCIENCE
  - PERCEPTS MAY NOT SUPPLY ALL RELEVANT INFORMATION
  - E.G., IN CARD GAME, DON'T KNOW CARDS OF OTHERS.

- RATIONAL IS DIFFERENT FROM BEING PERFECT
  - RATIONALITY MAXIMIZES EXPECTED OUTCOME WHILE PERFECTION MAXIMIZES ACTUAL OUTCOME.

AUTONOMY IN AGENTS

The **autonomy** of an agent is the extent to which its behaviour is determined by its own experience, rather than knowledge of designer.

- EXTREMES
  - NO AUTONOMY – IGNORES ENVIRONMENT/DATA
  - COMPLETE AUTONOMY – MUST ACT RANDOMLY/NO PROGRAM
- EXAMPLE: BABY LEARNING TO CRAWL
- IDEAL: DESIGN AGENTS TO HAVE SOME AUTONOMY
  - POSSIBLY BECOME MORE AUTONOMOUS WITH EXPERIENCE

PEAS

- PEAS: PERFORMANCE MEASURE, ENVIRONMENT, ACTUATORS, SENSORS
- MUST FIRST SPECIFY THE SETTING FOR INTELLIGENT AGENT DESIGN
- CONSIDER, E.G., THE TASK OF DESIGNING AN AUTOMATED TAXI DRIVER:
  - PERFORMANCE MEASURE: SAFE, FAST, LEGAL, COMFORTABLE TRIP, MAXIMIZE PROFITS
  - ENVIRONMENT: ROADS, OTHER TRAFFIC, PEDESTRIANS, CUSTOMERS
  - ACTUATORS: STEERING WHEEL, ACCELERATOR, BRAKE, SIGNAL, HORN
  - SENSORS: CAMERAS, SONAR, SPEEDOMETER, GPS, ODOMETER, ENGINE SENSORS, KEYBOARD

AHB

PEAS

- AGENT: PART-PICKING ROBOT
- PERFORMANCE MEASURE: PERCENTAGE OF PARTS IN CORRECT BINS
- ENVIRONMENT: CONVEYOR BELT WITH PARTS, BINS
- ACTUATORS: JOINTED ARM AND HAND
- SENSORS: CAMERA, JOINT ANGLE SENSORS

PEAS

- AGENT: INTERACTIVE ENGLISH TUTOR
- PERFORMANCE MEASURE: MAXIMIZE STUDENT'S SCORE ON TEST
- ENVIRONMENT: SET OF STUDENTS
- ACTUATORS: SCREEN DISPLAY (EXERCISES, SUGGESTIONS, CORRECTIONS)
- SENSORS: KEYBOARD

ENVIRONMENT TYPES

- FULLY OBSERVABLE (VS. PARTIALLY OBSERVABLE)
- DETERMINISTIC (VS. STOCHASTIC)
- EPISODIC (VS. SEQUENTIAL)
- STATIC (VS. DYNAMIC)
- DISCRETE (VS. CONTINUOUS)
- SINGLE AGENT (VS. MULTIAGENT):

FULLY OBSERVABLE (VS. PARTIALLY OBSERVABLE)

- **ACCESSIBLE (FULLY OBSERVABLE):**
- **THE AGENT HAS COMPLETE INFORMATION ABOUT THE ENVIRONMENT AT ALL TIMES.**
- **• EXAMPLE: CHESS GAME WHERE ALL PIECES AND THEIR POSITIONS ARE VISIBLE.**
- **• INACCESSIBLE (PARTIALLY OBSERVABLE):**
- **THE AGENT HAS INCOMPLETE OR NOISY INFORMATION ABOUT THE ENVIRONMENT.**
- **• EXAMPLE: A SELF-DRIVING CAR MAY NOT FULLY DETECT OBJECTS OBSCURED BY FOG.**

- IS EVERYTHING AN AGENT REQUIRES TO CHOOSE ITS ACTIONS AVAILABLE TO IT VIA ITS SENSORS? PERFECT OR FULL INFORMATION.
  - IF SO, THE ENVIRONMENT IS FULLY ACCESSIBLE
- IF NOT, PARTS OF THE ENVIRONMENT ARE INACCESSIBLE
  - AGENT MUST MAKE INFORMED GUESSES ABOUT WORLD.
- IN DECISION THEORY: PERFECT INFORMATION VS. IMPERFECT INFORMATION.

Cross Word	Poker	Backgammon	Taxi driver	Part picking robot	Image analysis
Fully	Partially	Partially	Partially	Fully	Fully

DETERMINISTIC (VS. STOCHASTIC)

- **DETERMINISTIC:**
- THE NEXT STATE OF THE ENVIRONMENT IS COMPLETELY DETERMINED BY THE CURRENT STATE AND THE AGENT'S ACTION.
- **EXAMPLE:** SOLVING A MATHEMATICAL PUZZLE.
- **STOCHASTIC:**
- THE ENVIRONMENT'S NEXT STATE INVOLVES RANDOMNESS OR UNCERTAINTY, REGARDLESS OF THE AGENT'S ACTION.
- **EXAMPLE:** ROLLING A DICE IN A BOARD GAME.

Cross Word	Poker	Backgammon	Taxi driver	Part picking robot	Image analysis
Deterministic	Stochastic	Stochastic	Stochastic	Stochastic	Deterministic

EPISODIC (VS. SEQUENTIAL):

- **EPISODIC:**
- EACH AGENT'S ACTION IS INDEPENDENT OF PREVIOUS ACTIONS. DECISIONS ARE MADE ON A PER-EPISODE BASIS.
- **EXAMPLE:** IMAGE RECOGNITION TASKS.
- **SEQUENTIAL:**
- THE AGENT'S ACTIONS INFLUENCE FUTURE DECISIONS OR OUTCOMES.
- **EXAMPLE:** PLAYING A VIDEO GAME WHERE PREVIOUS MOVES AFFECT THE CURRENT SCENARIO.

Cross Word	**Poker**	**Backgammon**	**Taxi driver**	**Part picking robot**	**Image analysis**
Sequential	Sequential	Sequential	Sequential	Episodic	Episodic

STATIC (VS. DYNAMIC):

- STATIC ENVIRONMENTS DON'T CHANGE
  - WHILE THE AGENT IS DELIBERATING OVER WHAT TO DO
- DYNAMIC ENVIRONMENTS DO CHANGE
  - SO AGENT SHOULD/COULD CONSULT THE WORLD WHEN CHOOSING ACTIONS
  - ALTERNATIVELY: ANTICIPATE THE CHANGE DURING DELIBERATION OR MAKE DECISION VERY FAST
- SEMIDYNAMIC: IF THE ENVIRONMENT ITSELF DOES NOT CHANGE WITH THE PASSAGE OF TIME BUT THE AGENT'S PERFORMANCE SCORE DOES.

Cross Word	**Poker**	**Backgammon**	**Taxi driver**	**Part picking robot**	**Image analysis**
Static	Static	Static	Dynamic	Dynamic	Semi

AHB

Another example: off-line route planning vs. on-board navigation system

DISCRETE (VS. CONTINUOUS)

- A LIMITED NUMBER OF DISTINCT, CLEARLY DEFINED PERCEPTS AND ACTIONS VS. A RANGE OF VALUES (CONTINUOUS)

Cross Word	**Poker**	**Backgammon**	**Taxi driver**	**Part picking robot**	**Image analysis**
Discrete	Discrete	Discrete	Conti	Conti	Conti

SINGLE AGENT (VS. MULTIAGENT):

- AN AGENT OPERATING BY ITSELF IN AN ENVIRONMENT OR THERE ARE MANY AGENTS WORKING TOGETHER

Cross Word	**Poker**	**Backgammon**	**Taxi driver**	**Part picking robot**	**Image analysis**
Single	Multi	Multi	Multi	Single	Single

Single-Agent Search Problems

Single-agent search problems involve a single entity (agent) attempting to navigate or solve a problem within an environment to achieve a specific goal. The agent's task is to determine an optimal sequence of actions to transition from an initial state to a goal state, often while minimizing costs or maximizing rewards.

AI agents are central to building intelligent systems capable of interacting with, learning from, and influencing their environments. By categorizing agents based on their capabilities and the environments they operate in, developers can create systems that range from simple reactive agents to complex, learning-based, and goal-driven systems. As AI continues to advance, the sophistication of these agents will also increase, allowing for more adaptive, efficient, and intelligent solutions to real-world problems.

Key Components of Single-Agent Search Problems

1. **State Space**:

• The set of all possible configurations of the environment.

• Example: In a maze, states could represent the positions of the agent.

2. **Initial State**:

• The starting point of the agent in the search process.

• Example: The top-left corner of a maze.

3. **Actions**:

• The set of actions the agent can take to transition between states.

• Example: Move up, move down, move left, move right.

4. **Transition Model**:

• Defines how actions affect the state. It specifies the result of taking a specific action in a given state.

• Example: Moving right from (x, y) transitions to $(x+1, y)$.

5. **Goal State**:

• The desired configuration or state the agent must reach.

• Example: The bottom-right corner of the maze.

6. **Path Cost**:

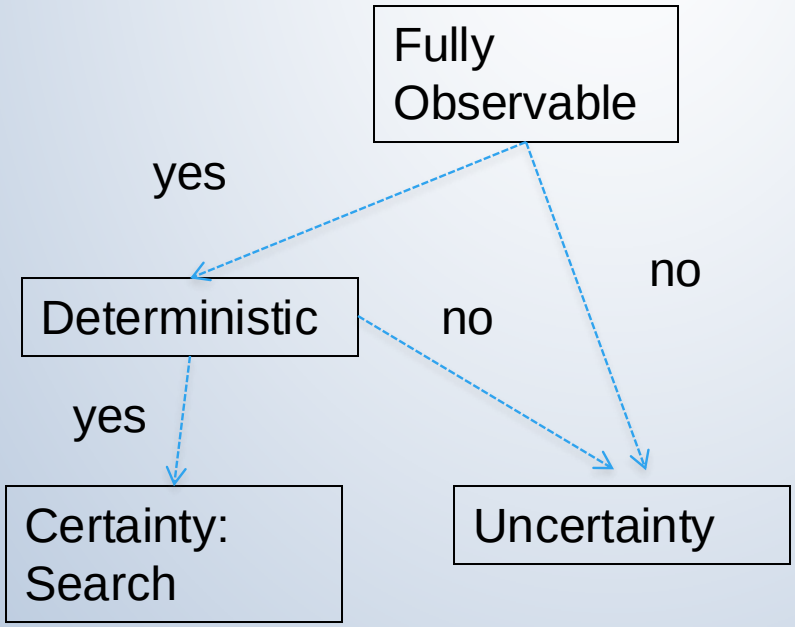
• A numeric value that accumulates the cost of actions taken to transition between states.

• Example: Each step in a maze might cost 1 unit, or some actions may have higher costs.

SUMMARY.

	Observable	Deterministic	Episodic	Static	Discrete	Agents
Cross Word	Fully	Deterministic	Sequential	Static	Discrete	Single
Poker	Fully	Stochastic	Sequential	Static	Discrete	Multi
Backgammon	Partially	Stochastic	Sequential	Static	Discrete	Multi
Taxi driver	Partially	Stochastic	Sequential	Dynamic	Conti	Multi
Part picking robot	Partially	Stochastic	Episodic	Dynamic	Conti	Single
Image analysis	Fully	Deterministic	Episodic	Semi	Conti	Single

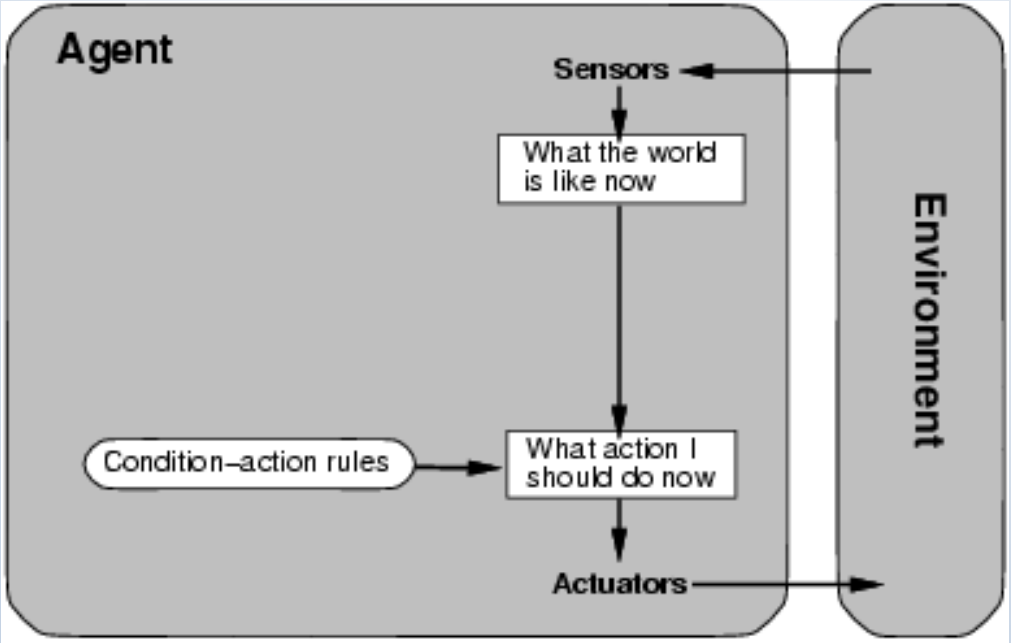
CHOICE UNDER (UN)CERTAINTY



AGENT TYPES

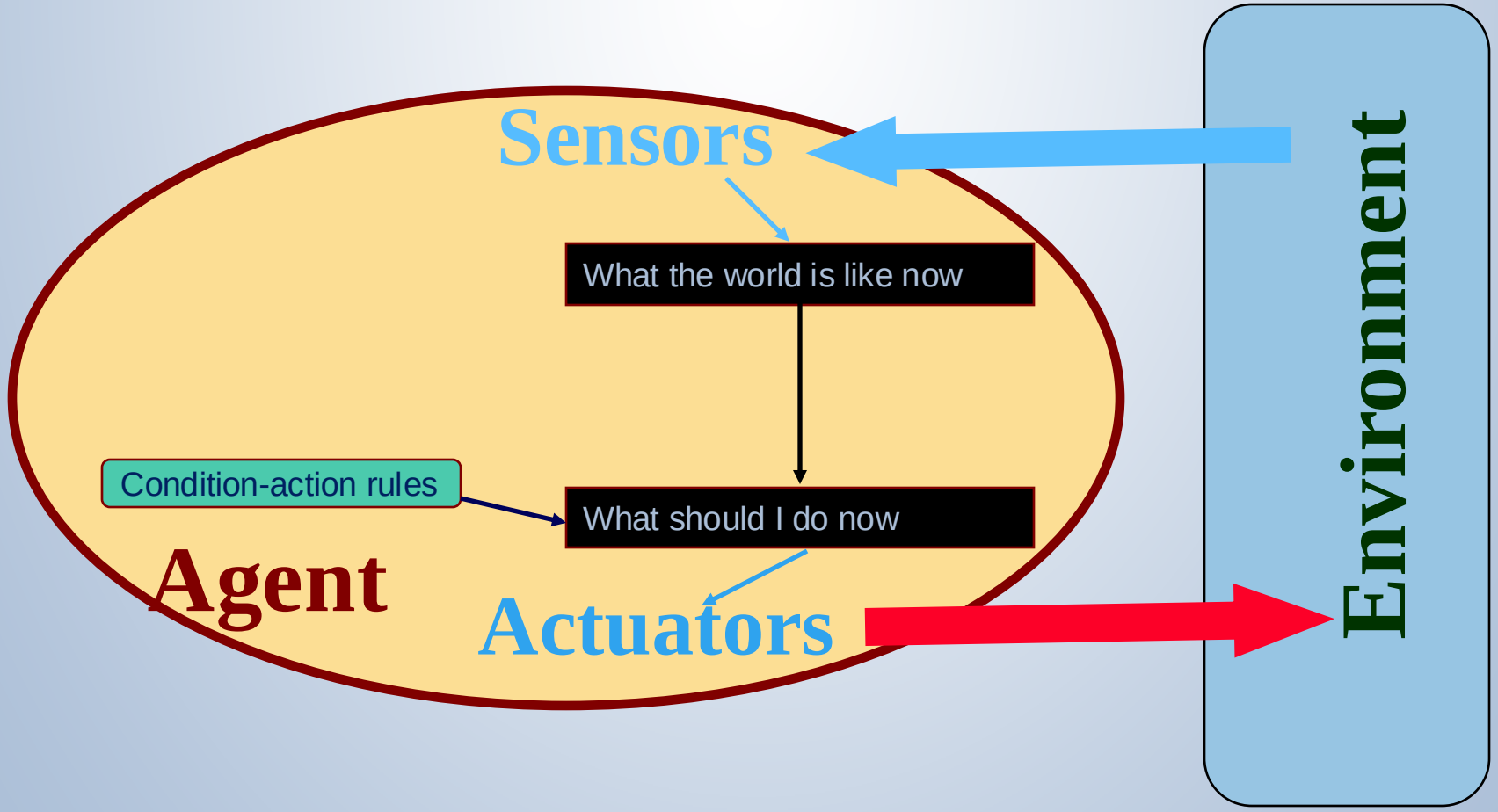
- FOUR BASIC TYPES IN ORDER OF INCREASING GENERALITY:
  - SIMPLE REFLEX AGENTS
  - REFLEX AGENTS WITH STATE/MODEL
  - GOAL-BASED AGENTS
  - UTILITY-BASED AGENTS
  - ALL THESE CAN BE TURNED INTO LEARNING AGENTS
  - HTTP://WWW.AI.SRI.COM/~OREILLY/AIMA3EJAVA/AIMA3EJAVADEMOS.HTML

SIMPLE REFLEX AGENTS




```
function REFLEX-VACUUM-AGENT( [location,status]) returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

SIMPLE REFLEX AGENTS



SIMPLE REFLEX AGENTS

- SIMPLE BUT VERY LIMITED INTELLIGENCE.
- **ACTION DOES NOT DEPEND ON PERCEPT HISTORY, ONLY ON CURRENT PERCEPT.**
- THEREFORE NO MEMORY REQUIREMENTS.
- INFINITE LOOPS
  - SUPPOSE VACUUM CLEANER DOES NOT OBSERVE LOCATION. WHAT DO YOU DO GIVEN LOCATION = CLEAN? LEFT OF A OR RIGHT ON B -> INFINITE LOOP.
  - FLY BUZZING AROUND WINDOW OR LIGHT.
  - POSSIBLE SOLUTION: RANDOMIZE ACTION.
  - THERMOSTAT.
- CHESS – OPENINGS, ENDINGS
  - LOOKUP TABLE (NOT A GOOD IDEA IN GENERAL)
    - 35^{100} ENTRIES REQUIRED FOR THE ENTIRE GAME

STATES: BEYOND REFLEXES

- RECALL THE AGENT FUNCTION THAT MAPS FROM PERCEPT HISTORIES TO ACTIONS:

$$[F: \mathcal{P}^{\star} \rightarrow \mathcal{A}]$$

- AN AGENT PROGRAM CAN IMPLEMENT AN AGENT FUNCTION BY MAINTAINING AN **INTERNAL STATE**.
- THE INTERNAL STATE CAN CONTAIN INFORMATION ABOUT THE STATE OF THE EXTERNAL ENVIRONMENT.
- THE STATE DEPENDS ON THE HISTORY OF PERCEPTS AND ON THE HISTORY OF ACTIONS TAKEN:

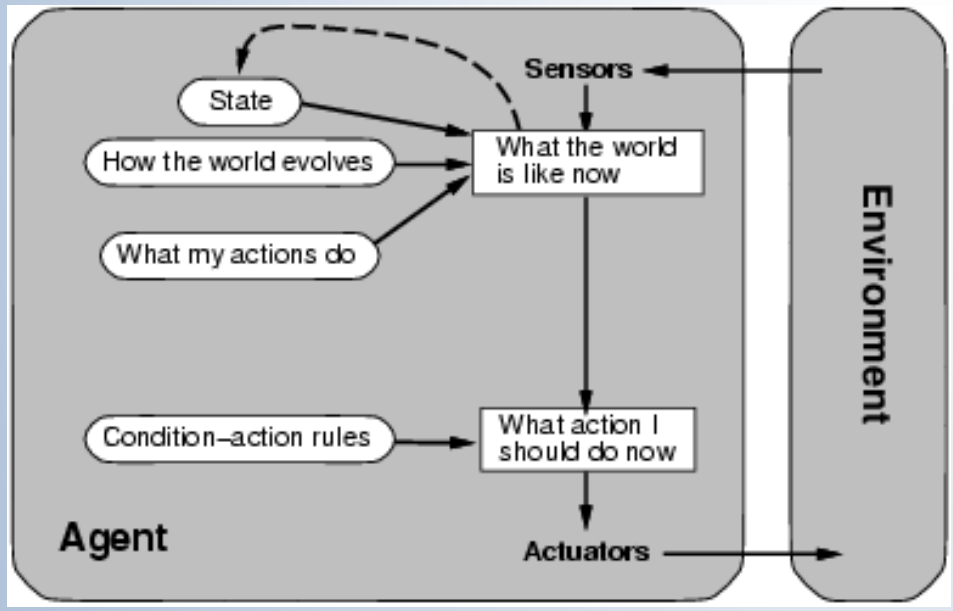
$[F: \mathcal{P}^{\star}, \mathcal{A}^{\star} \rightarrow \mathcal{S} \rightarrow \mathcal{A}]$ WHERE $\mathcal{S}$ IS THE SET OF STATES.

- IF EACH INTERNAL STATE INCLUDES ALL INFORMATION RELEVANT TO INFORMATION MAKING, THE STATE SPACE IS **MARKOVIAN**.

STATES AND MEMORY: GAME THEORY

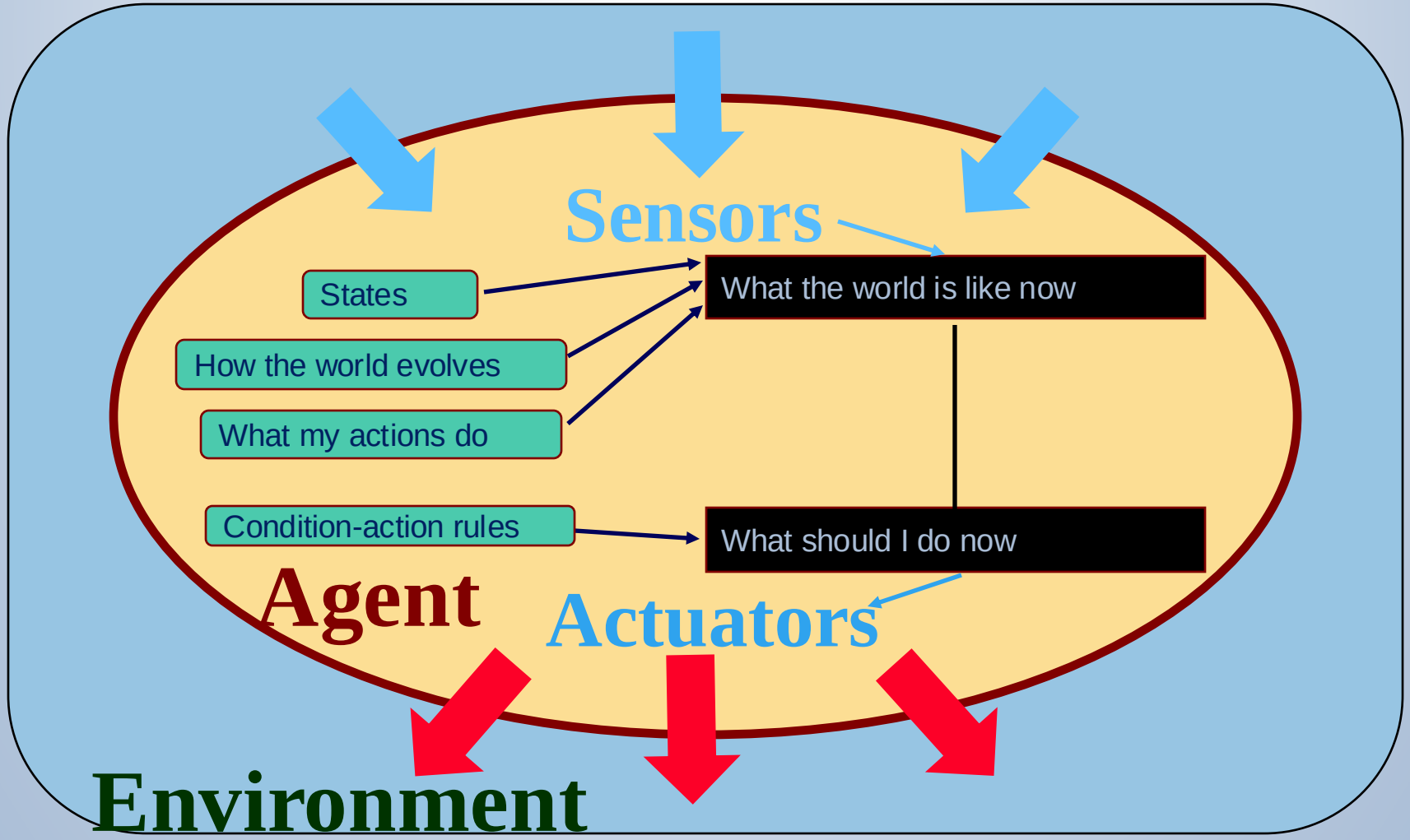
- IF EACH STATE INCLUDES THE INFORMATION ABOUT THE PERCEPTS AND ACTIONS THAT LED TO IT, THE STATE SPACE HAS **PERFECT RECALL**.
- **PERFECT INFORMATION** = PERFECT RECALL + FULL OBSERVABILITY + DETERMINISTIC ACTIONS.

MODEL-BASED REFLEX AGENTS



- Know how world evolves
  - Overtaking car gets closer from behind
- How agents actions affect the world
  - Wheel turned clockwise takes you right
- Model base agents update their state

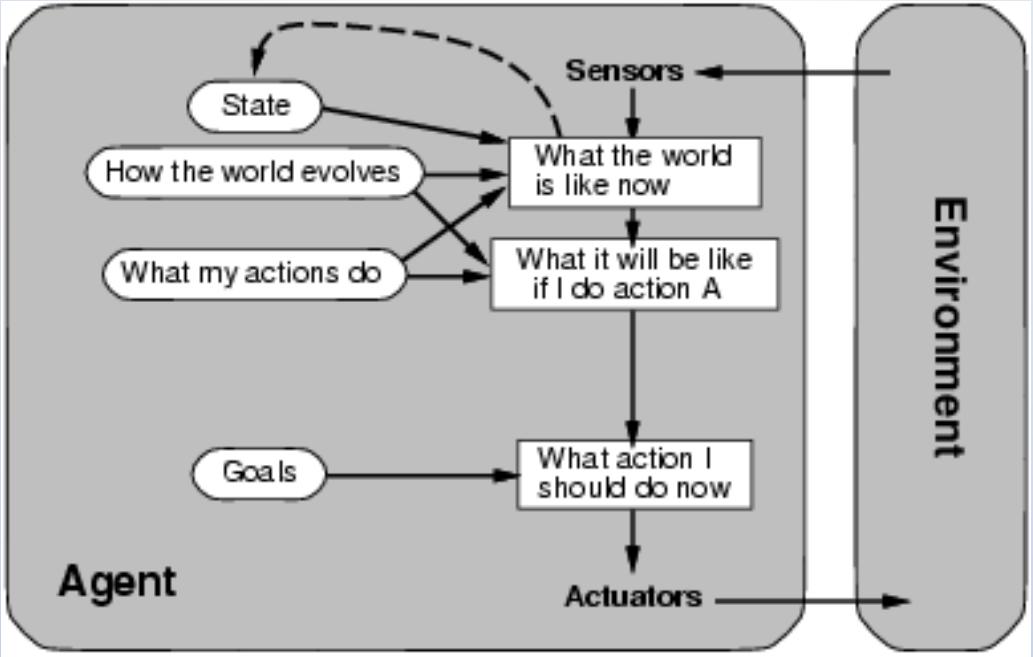
```
function REFLEX-AGENT-WITH-STATE(percept) returns action
  static: state, a description of the current world state
          rules, a set of condition-action rules

  state ← UPDATE-STATE(state, percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  state ← UPDATE-STATE(state, action)
  return action
```


Sensors
States
How the world evolves
What my actions do
What the world is like now
Condition-action rules
What should I do now
Agent
Actuators
Environment

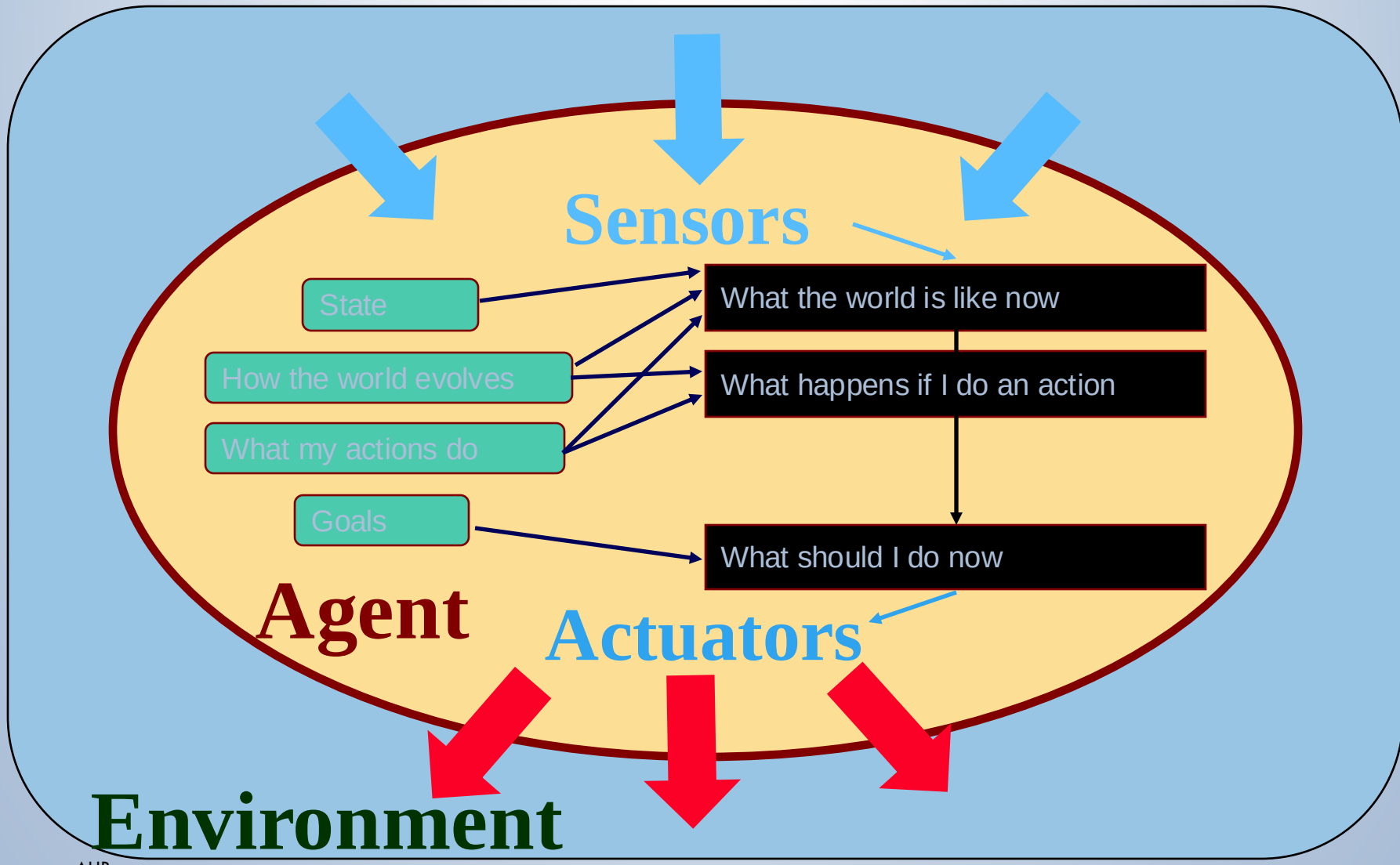
GOAL-BASED AGENTS

- KNOWING STATE AND ENVIRONMENT? ENOUGH?
  - TAXI CAN GO LEFT, RIGHT, STRAIGHT
- HAVE A GOAL
  - A DESTINATION TO GET TO
- USES KNOWLEDGE ABOUT A GOAL TO GUIDE ITS ACTIONS
  - E.G., SEARCH, PLANNING

GOAL-BASED AGENTS



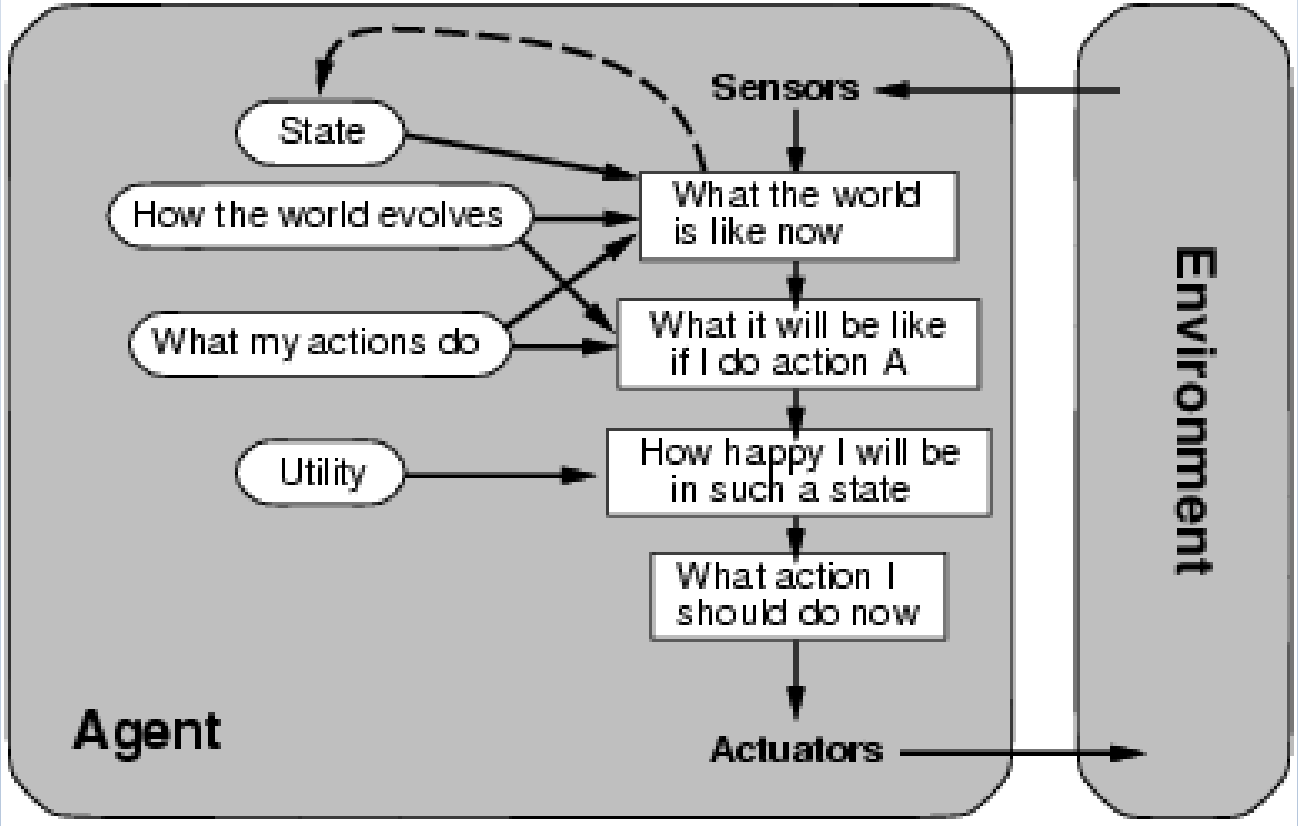
- Reflex agent breaks when it sees brake lights. Goal based agent reasons
  - Brake light -> car in front is stopping -> I should stop -> I should use brake


Sensors
State
How the world evolves
What my actions do
Goals
What the world is like now
What happens if I do an action
What should I do now
Agent
Actuators
Environment

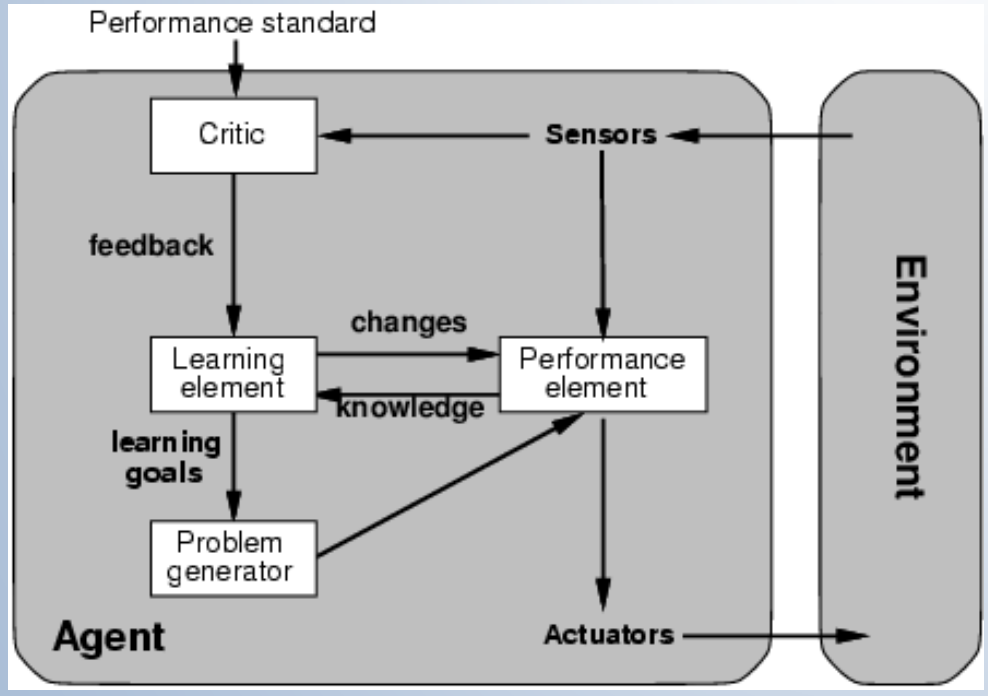
UTILITY-BASED AGENTS

- GOALS ARE NOT ALWAYS ENOUGH
  - MANY ACTION SEQUENCES GET TAXI TO DESTINATION
  - CONSIDER OTHER THINGS. HOW FAST, HOW SAFE…..
- A UTILITY FUNCTION MAPS A STATE ONTO A REAL NUMBER WHICH DESCRIBES THE ASSOCIATED DEGREE OF "HAPPINESS", "GOODNESS", "SUCCESS".
- WHERE DOES THE UTILITY MEASURE COME FROM?
  - ECONOMICS: MONEY.
  - BIOLOGY: NUMBER OF OFFSPRING.
  - YOUR LIFE?

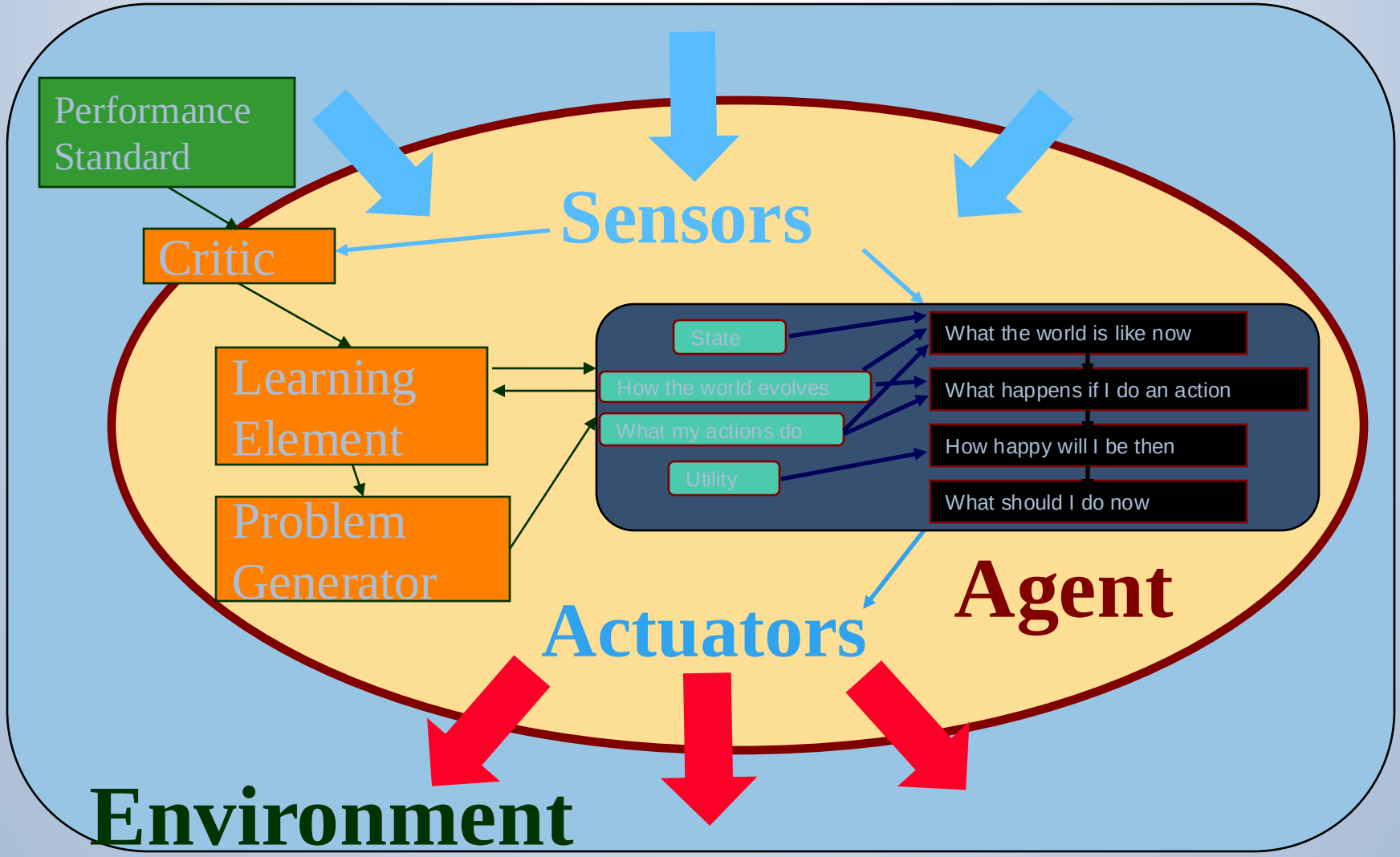
UTILITY-BASED AGENTS



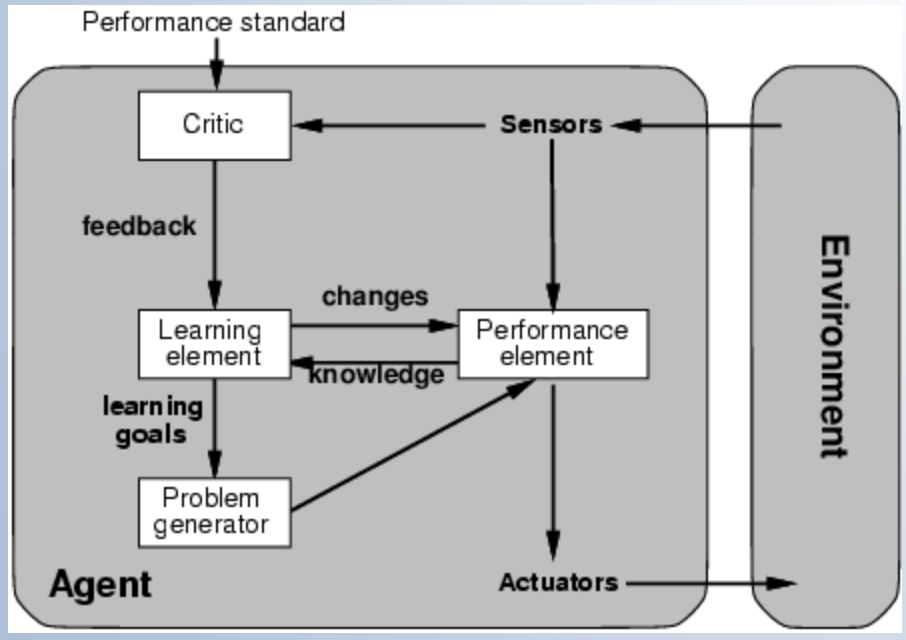
LEARNING AGENTS



- Performance element is what was previously the whole agent
  - Input sensor
  - Output action
- Learning element
  - Modifies performance element.


Performance Standard
Critic
Learning Element
Problem Generator
Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What happens if I do an action
How happy will I be then
What should I do now
Agent
Actuators
Environment

LEARNING AGENTS

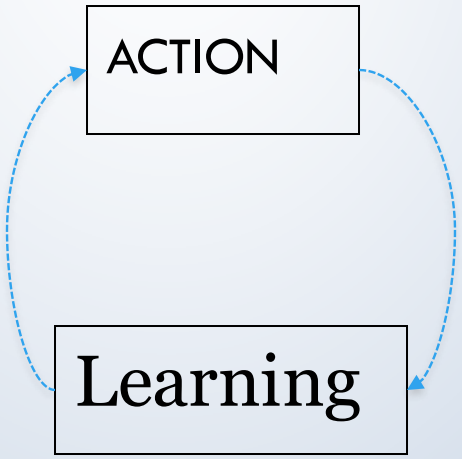


- Critic: how the agent is doing
  - Input: checkmate?
  - Fixed
- Problem generator
  - Tries to solve the problem differently instead of optimizing.
  - Suggests **exploring** new actions -> new problems.

LEARNING AGENTS(TAXI DRIVER)

- PERFORMANCE ELEMENT
  - HOW IT CURRENTLY DRIVES
- TAXI DRIVER MAKES QUICK LEFT TURN ACROSS 3 LANES
  - CRITICS OBSERVE SHOCKING LANGUAGE BY PASSENGER AND OTHER DRIVERS AND INFORMS BAD ACTION
  - LEARNING ELEMENT TRIES TO MODIFY PERFORMANCE ELEMENTS FOR FUTURE
  - PROBLEM GENERATOR SUGGESTS EXPERIMENT OUT SOMETHING CALLED BRAKES ON DIFFERENT ROAD CONDITIONS
- EXPLORATION VS. EXPLOITATION
  - LEARNING EXPERIENCE CAN BE COSTLY IN THE SHORT RUN
  - SHOCKING LANGUAGE FROM OTHER DRIVERS
  - LESS TIP
  - FEWER PASSENGERS

THE PICTURE FOR REFLEX-BASED AGENTS



Reinforcement Learning

- Studied in AI, Cybernetics, Control Theory, Biology, Psychology.

Search Strategies are crucial for solving problems by navigating through various possible states or configurations to reach a desired goal. These strategies can be categorized into **uninformed (blind)** and **informed (heuristic-based)** search strategies, with each type applicable to different kinds of problem-solving scenarios.

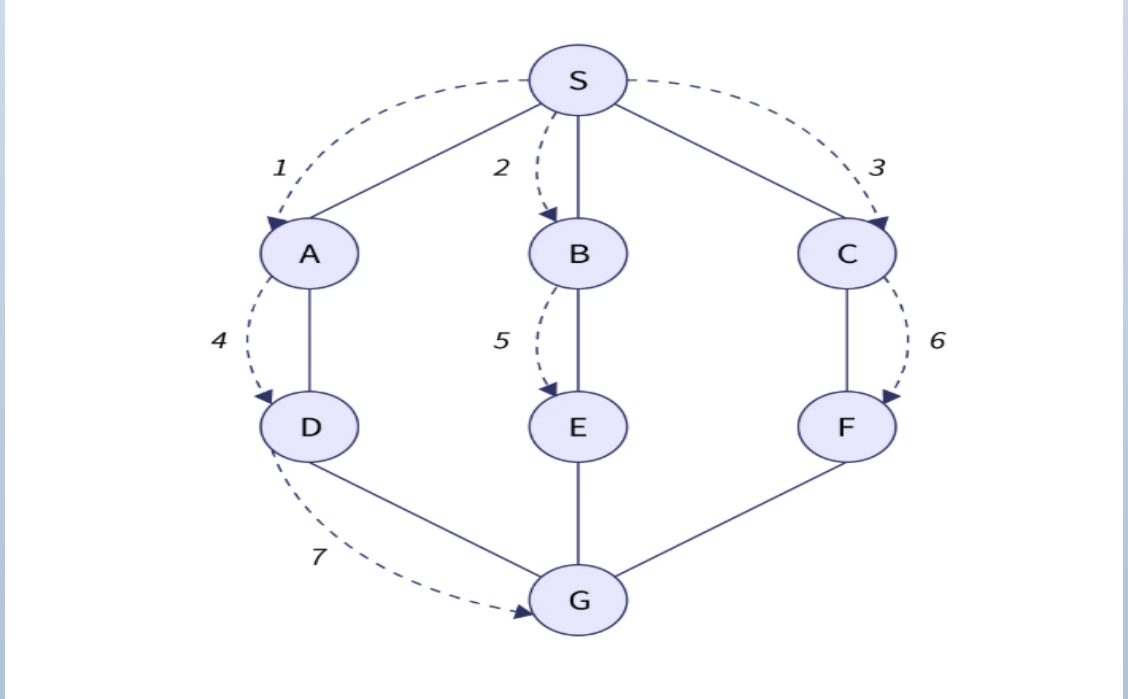
1. Uninformed Search Strategies (Blind Search)

Uninformed search strategies do not have any additional information about the goal state beyond the problem definition itself. These strategies systematically explore the search space.

Breadth-First Search (BFS)

Breadth-First Search (BFS), which starts at a root node, explores all neighbors at the current depth, then proceeds to the next depth, using a queue for tracking. It's ideal for finding shortest paths in unweighted graphs.

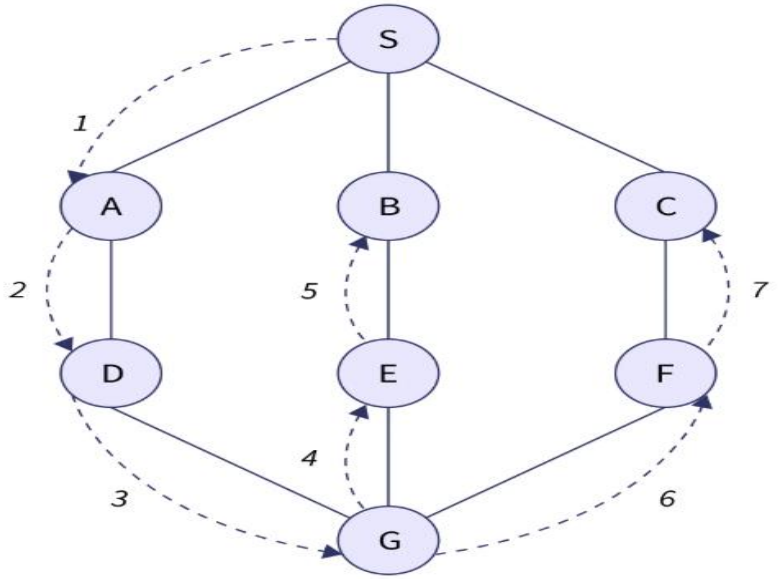
- **Advantages**: Guarantees finding the shortest path to the goal in an unweighted graph.
- **Disadvantages**: BFS is memory-intensive as it needs to store all nodes at the current depth.



Depth-First Search (DFS)

DFS(Depth First Search) is an edge-based traversal method that explores vertices recursively, utilizing a stack for storing visited vertices in a LIFO manner. It's faster and more memory-efficient than BFS. By prioritizing deep unexplored nodes and backtracking as it pops elements from the stack, DFS ensures each vertex is visited exactly once and each edge is checked twice.

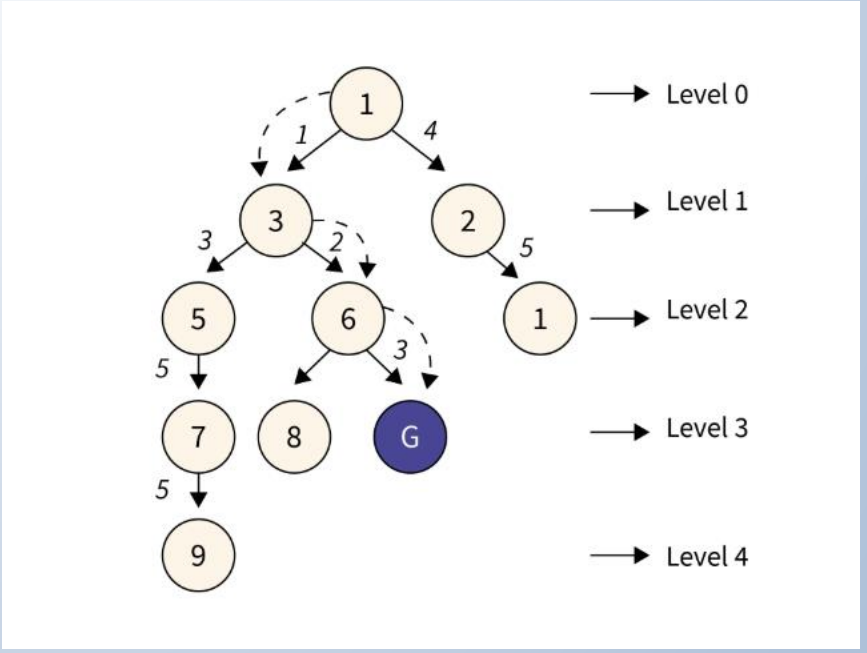
- **Advantages:** Requires less memory than BFS and can be easily implemented using recursion**.**
- **Disadvantages**: DFS can get stuck in cycles or infinite branches unless a depth limit is set, and it does not guarantee the shortest path.



Uniform-Cost Search Algorithm

The Uniform Cost Search Algorithm is a search algorithm to find the minimum cumulative cost of the path from the source node to the destination node. It is an uninformed algorithm i.e. it doesn't have prior information about the path or nodes and that is why it is a brute-force approach.

- UCS is a variant of BFS that takes into account the cost of each edge. It expands the least-cost node first and is guaranteed to find the lowest-cost path to the goal.
- **Advantages**: Guarantees finding the least-cost path.
- **Disadvantages**: UCS can be slow, especially if all edge costs are the same (degenerating into BFS).



Depth-Limited Search

This is a variation of DFS that limits the depth of the search to a predefined limit, preventing infinite recursion in problems with infinite depth spaces.

- **Advantages**: Solves the problem of infinite loops in DFS.
- **Disadvantages**: The solution may not be optimal if the depth limit is too small.

Iterative Deepening Search (IDS)

- IDS combines the benefits of DFS and BFS by gradually increasing the depth limit in each iteration. It performs a DFS up to a depth limit, then increases the depth and repeats. IDDFS calls DFS for different depths starting from an initial value. In every call, DFS is restricted from going beyond given depth. So basically we do DFS in a BFS fashion.
- Iterative deepening depth-first search is a hybrid algorithm emerging out of BFS and DFS. IDDFS might not be used directly in many applications of Computer Science, yet the strategy is used in searching data of infinite space by incrementing the depth limit by progressing iteratively. This is quite useful and has applications in AI and the emerging data sciences industry.
- **Advantages**: Guarantees the optimal solution while using less memory than BFS.
- **Disadvantages**: It involves repeated exploration of nodes, which can lead to inefficiency.

Thank You