INTRODUCTION- INTELLIGENT AGENTS

Chapter 1

AGENDA-CHAPTER 1

What is AI?

Foundation of AI,

State of Art

Agents of Environment

Structure of Agents

WHAT IS AI?

“[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning..” (Bellman, 1978)	“The study of the computations that make it possible to perceive, reason and act” (Winston, 1992)
“The study of how to make computers do things at which, at the moment, people are better” (Rich+Knight, 1991)	“The branch of computer science that is concerned with the automation of intelligent behaviour” (Luger+Stubblefield, 1993)

Views of AI fall into four categories:

Thinking Humanly	Thinking rationally
Acting Humanly	Acting rationally

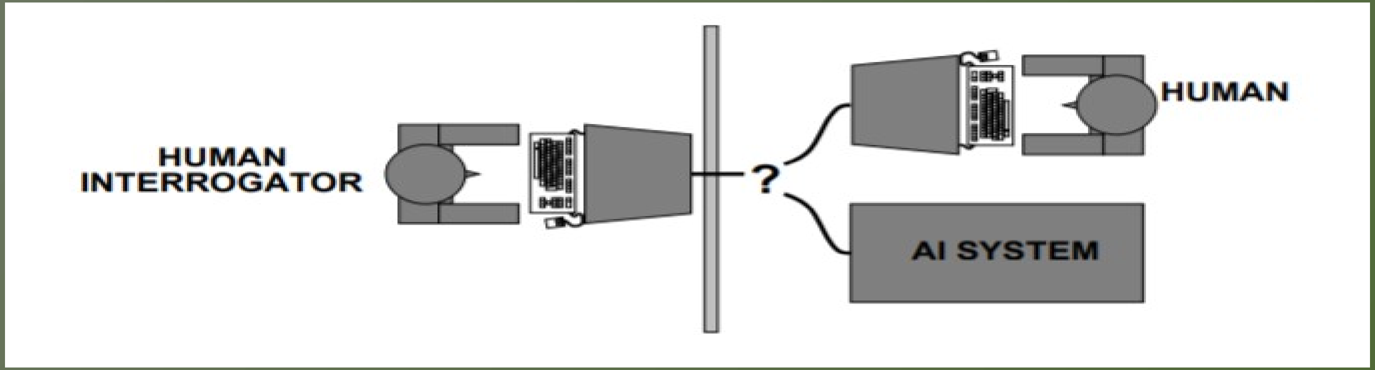
Slides by Disha D N

ACTING HUMANLY: THE TURING TEST

Turing (1950) “Computing machinery and intelligence”

Can machine think?----◊ “Can machines behave intelligently?”

“When AI system tries to communicate with the human interrogator in written format, for some time if human interrogator confused with whom he is interacting with”-AI system actually acts like human



ACTING HUMANLY: THE TURING TEST

In order to act like human what all components AI system should possess?

Suggested major components of AI: knowledge, reasoning, language, understanding, learning

Problem: Turing test is not reproducible, constructive, or amenable to mathematical analysis

The implications for the Turing Test are clear: The ability to provide good answers to human questions does not necessarily imply that the provider of those answers is thinking; passing the Test is no proof of active intelligence.

THINKING HUMANLY: COGNITIVE SCIENCE

If we say that the given problem thinks like a human we must have some way of determining how human think?. We need to get inside the actual working of human minds.

There are three ways to do this: through introspection- trying to catch our own thoughts as they go by.

Through psychological experiments: observing a person in action

Through Brain imaging: observing a brain in action

Once we have a sufficiently precise theory of the mind then we can express the theory as a computer.

If the program’s input output behaviour matches corresponding human behaviour, that is the evidence that some of the program’s mechanisms could also be operating in humans.

THINKING RATIONALLY: LAWS OF THOUGHT

Always think correctly.

It is based on some laws or inferences of action

It should always thinks correctly hence it requires 100% knowledge

It requires too many computations

ACTING RATIONALLY-RATIONAL AGENT APPROACH

Acting Rationally means: doing the things rightly and behaving rightly.

It is the generalized approach.

We use rational agents here. Agent is one with sensors and actuators

Agents are always tries to maximise the performance for a given information.

Advantages: It is more general than laws of thought approach, because correct inference is just one of several possible mechanisms for achieving rationality.

It is more amenable to scientific development

Problem is it is not always possible to do right things in complicated environment.

STATE OF ARTS

What AI can do today? What all the fields it can include?

Robotics : Self driving cars, Vacuum cleaners

Speech Recognition : Any reservations can be booked with automatic speech recognition and dialog management system

Autonomous planning and scheduling: NASA’s autonomous planning agent can operate the spacecraft

Game playing: IBM’s DEEP BLUE became the first computer program to defeat the world champion in a chess match

STATE OF ARTS

Spam fighting: Classifications of messages and calls as spams

Logistic planning: Ex: DART (Dynamic Analysis and Replanning tool) to do automated logistics planning, scheduling for transportation

Machine Translation: Ex: Computer program automatically translates from Arabic to English

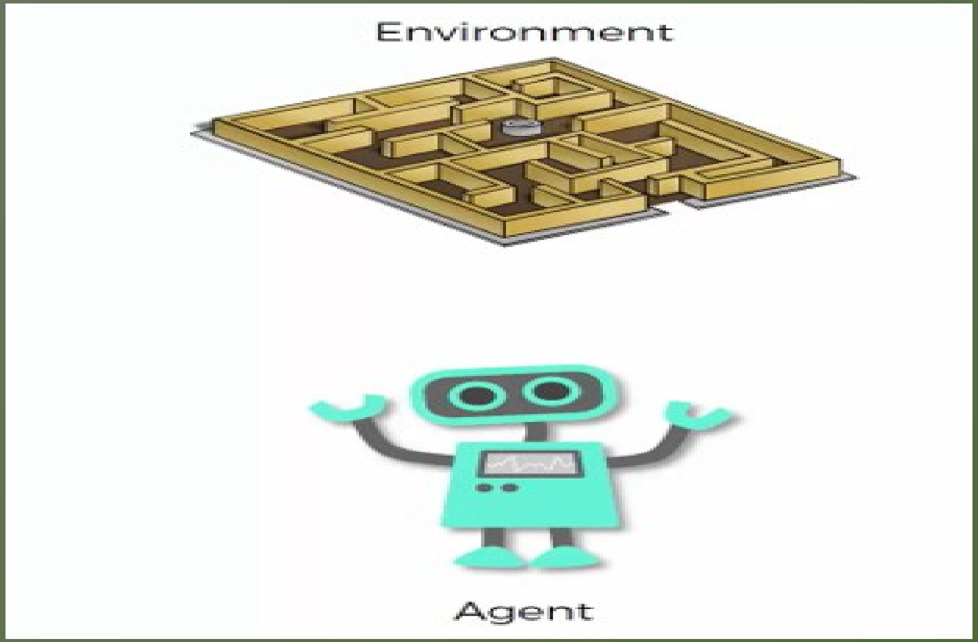
CHAPTER-2

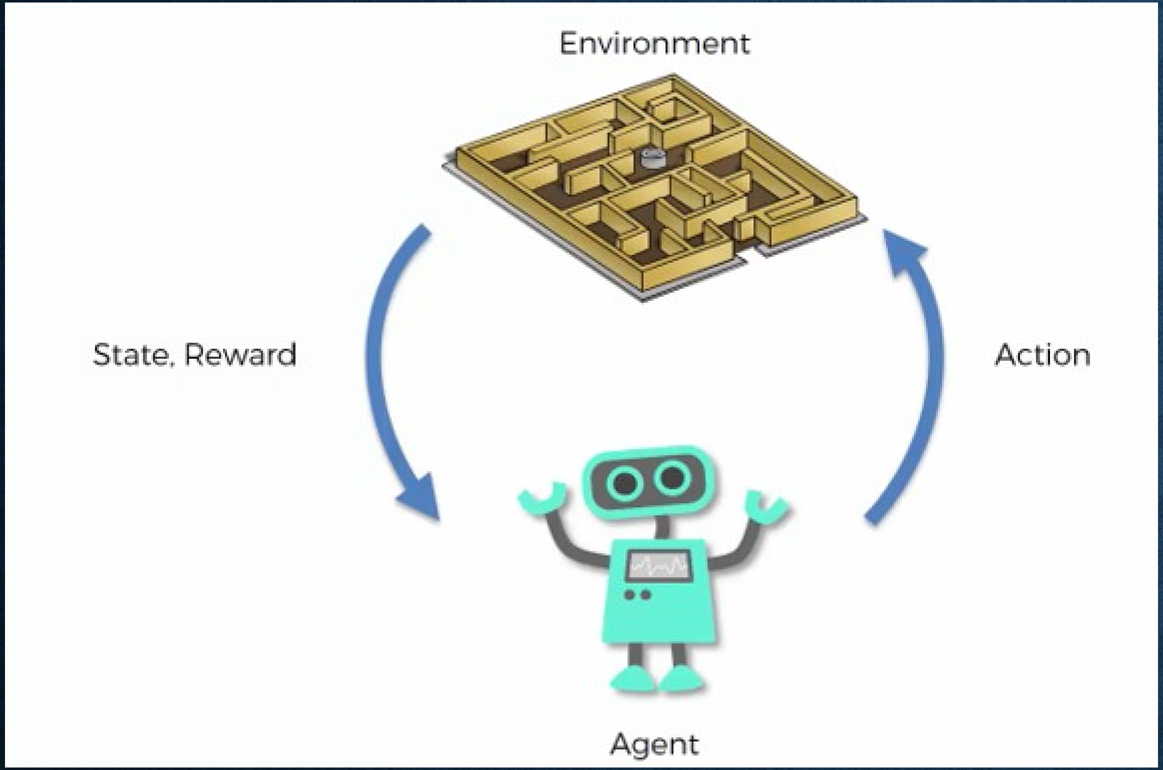
Intelligent Agents

Slides by Disha D N

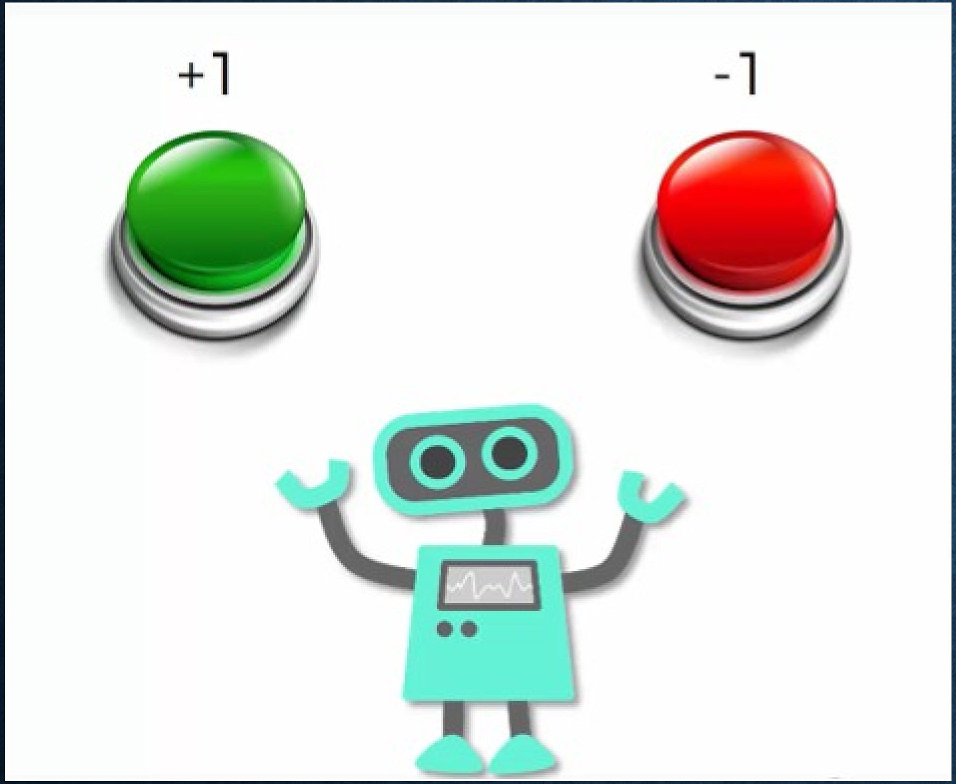
WHAT IS AN AGENT?

An agent is something that acts. They are expected to do more; operate autonomously, perceive their environment, persist over prolonged time period, adapt to change and create and perceive goals.



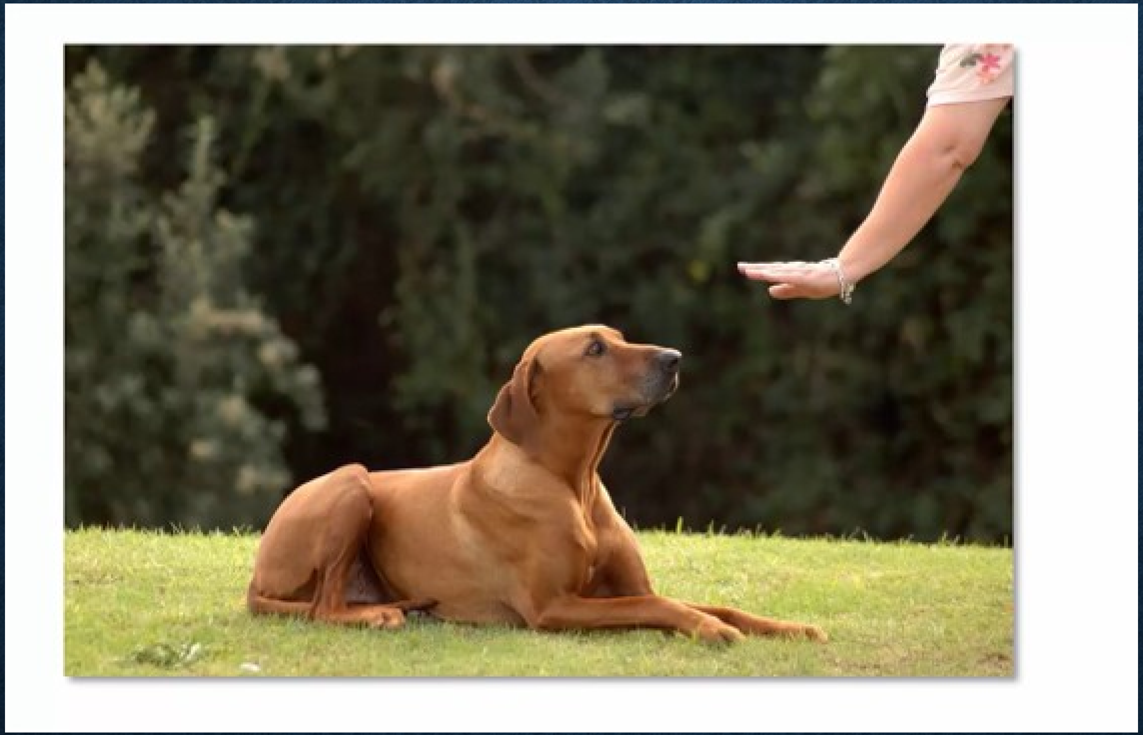


Agent will observe environment
Perform some action
He will explore different state
He will be awarded.



Rewards Can be Positive or it can be negative.
It depends on the actions performed by the agent

AGENT CAN BE ANYTHING??



Slides by Disha D N

EXAMPLE OF AGENT WITH AI

RATIONAL AGENTS

Rational Decision: For problem solving, if an agent makes a decision based on some logical reasoning, then the decision is called as “Rational Decision”

It is similar to the way how human have the ability to make right decisions, based on his/her experience and logical reasoning.

Similarly, an agent should also be able to make correct decisions based on what is known from the percept sequence and actions which are carried out by that agent from its knowledge-based on past experience

RATIONAL AGENTS

Agents perceive their environment through sensors over a prolonged time period and adapt to change to create and pursue goals and take actions through actuators to achieve those goals.

Rational agent is the one that does right things and acts rationally so as to achieve the best outcome even when there is uncertainty in knowledge

A rational agent can be anything that makes a decisions, typically a person, machine or a software program

RATIONAL AGENTS

Abstractly, an agent is a function from percept histories to actions:

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

For any given class of environments and tasks, we seek the agents (or class of agents) with the best performance

AGENTS AND ENVIRONMENT

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

Ex 1: Human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract and so on for actuators.

Ex 2: A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.

Ex 3: A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files and sending network packets.

AGENT'S INTERACTION

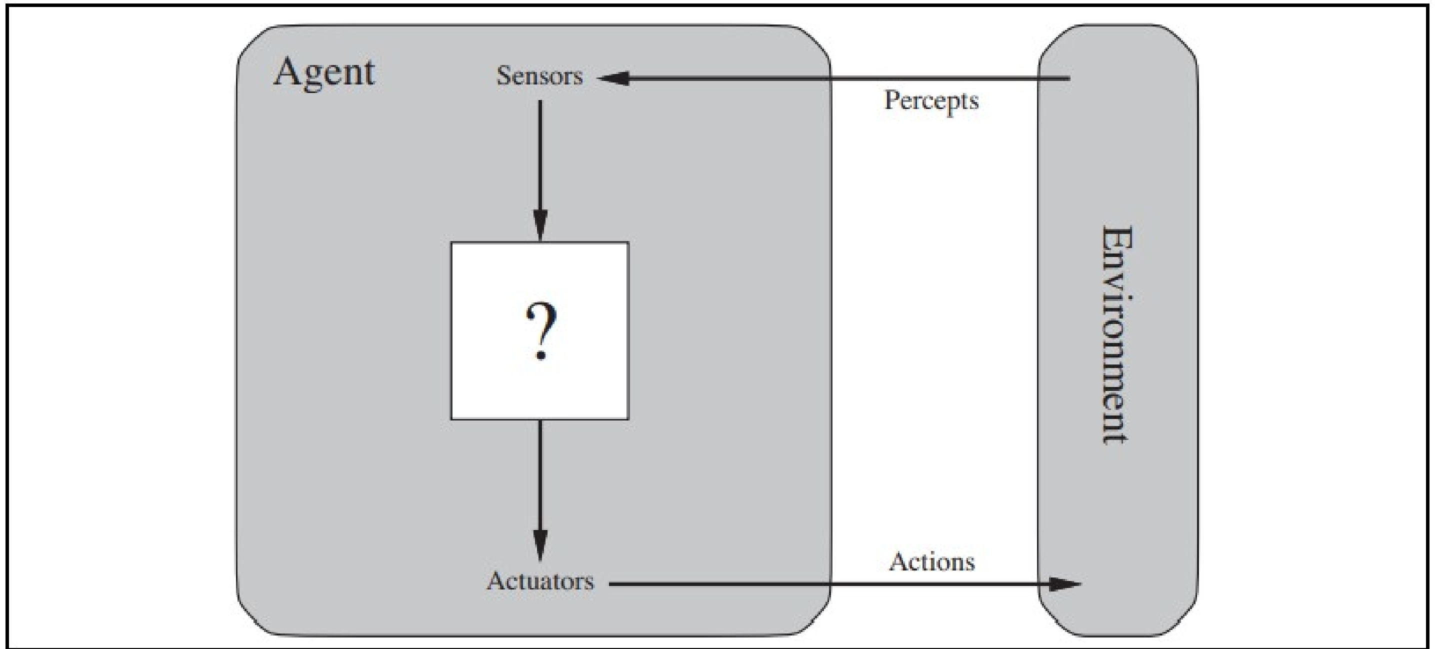


Figure 2.1 Agents interact with environments through sensors and actuators.

Slides by Disha D N

We use the term percept to refer to the agent’s perceptual inputs at any given instant.

An agent’s percept sequence is the complete history of everything the agent has ever perceived.

Agent’s choice of action at any given instant can depend on the entire percept sequence observed to the date but not on anything that it hasn’t perceived.

SIMPLE AGENT EXAMPLE

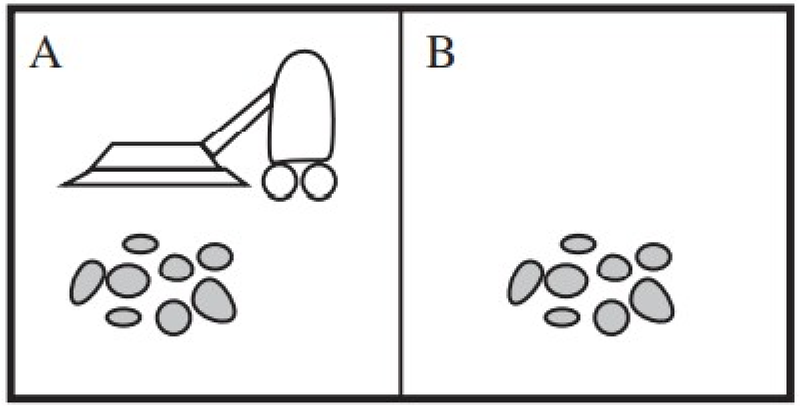


Figure 2.2 A vacuum-cleaner world with just two locations.

Consider a vacuum cleaner world as shown in figure. Here we defined only two locations, squares A and B.

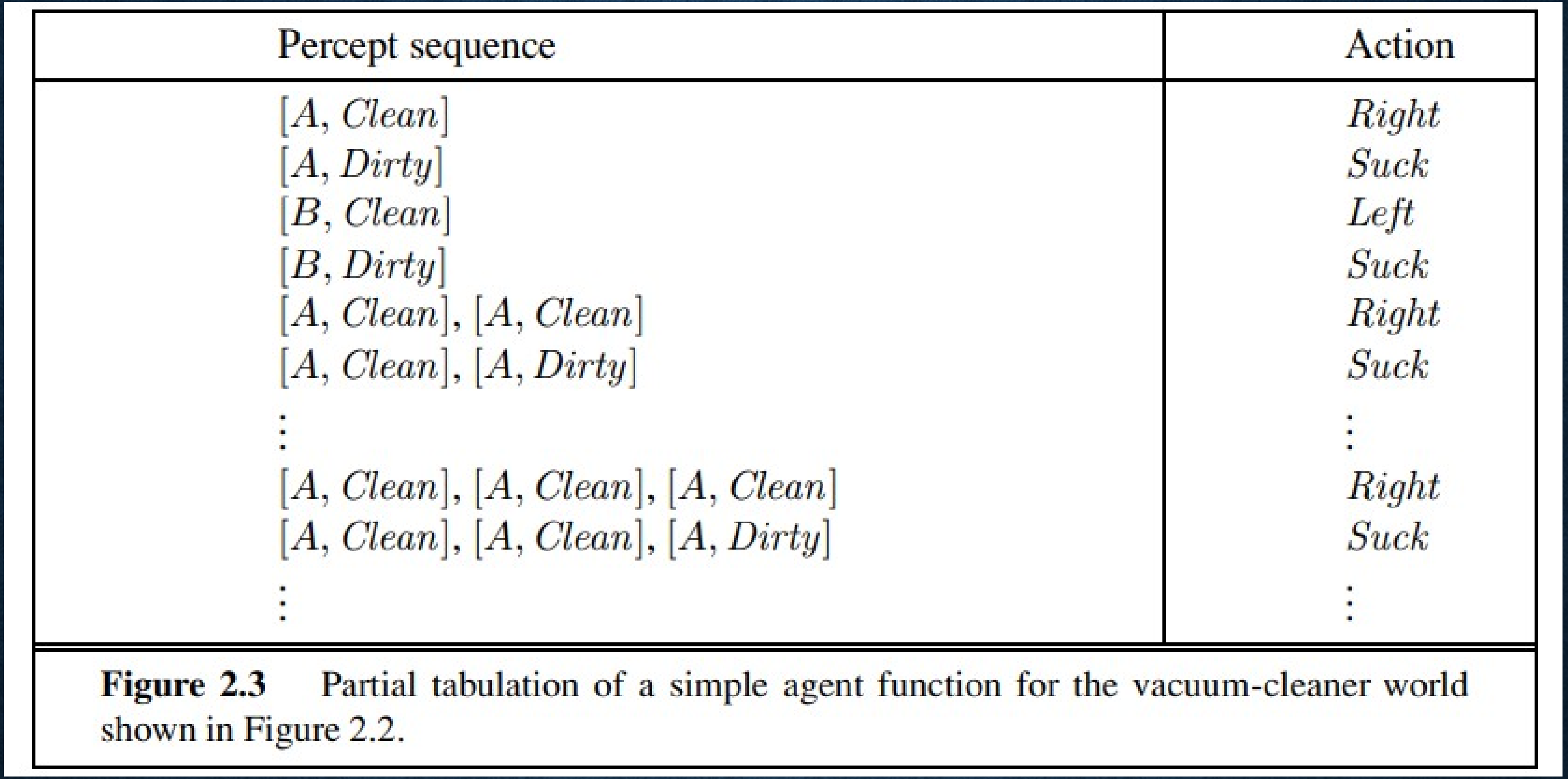
The vacuum agent perceives which square it is in and whether there is dirt in the square.

It can choose to move left, move right, suck up the dirt or do nothing.

One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square.

A partial tabulation of the square is as shown in the figure with implementation.

Note: “Even for simple agent like vacuum cleaner with only two squares as the environment we must be able to tell what actions results in good behaviour or how can you say that the agent behaves correctly ??”



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

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2.

GOOD BEHAVIOUR: THE CONCEPT OF RATIONALITY

Rational agent is one that does right thing-conceptually speaking, every entry in the table for agent is filled out correctly.

But what does this right thing mean? Or how you measure the right thing?

<u>Rationality: we measure the right thing through rationality</u>

Rationality at any given time depends on four things:

- Performance measure that defines the criterion of success
- The agent’s prior knowledge of the environment
- The actions that agent can perform
- The agent’s percept to date

This leads to the definition of rational agent

VACUUM CLEANER IS RATIONAL AGENT OR NOT?

It depends!!

We need to say what performance measure is, what is known about the environment, what sensors and actuators agent has?

Let us put all these now

- Performance measure awards one point for each clean square at each time step, over a lifetime of 1000 time step
- The geography of environment is known priori but the dirt distribution and initial location of the agent is not known
- The only available actions are Left, Right, Suck
- The agent correctly perceives its location and whether that location contains dirt

Under these circumstances we can say that vacuum cleaner is a rational agent

VACUUM CLEANER, IS IT IRRATIONAL?

It is possible to prove that same agent is irrational….!! How?

Once all the dirt are cleaned up, the agent may oscillate needlessly back and forth;

Hence we need to introduce some penalty of one point for each movement left or right.

A better agent is do nothing once it is sure that all the squares are clean.

But will it stop exploring?!!

If clean squares can become dirty again, agent should occasionally check and clean them.

OMNISCIENCE, LEARNING, AND AUTONOMY

An omniscient agent knows actual outcome of its actions and can act accordingly; but omniscience is impossible in reality.

Scenario

"Consider the following example: I am walking along the Champs Elys´ees one day and I see an old friend across the street. There is no traffic nearby and I'm not otherwise engaged, so, being rational, I start to cross the street. Meanwhile, at 33,000 feet, a cargo door falls off a passing airliner,2 and before I make it to the other side of the street I am flattened. Was I irrational to cross the street? It is unlikely that my obituary would read "Idiot attempts to cross street."

THE NATURE OF ENVIRONMENT

What we are going to study in this??

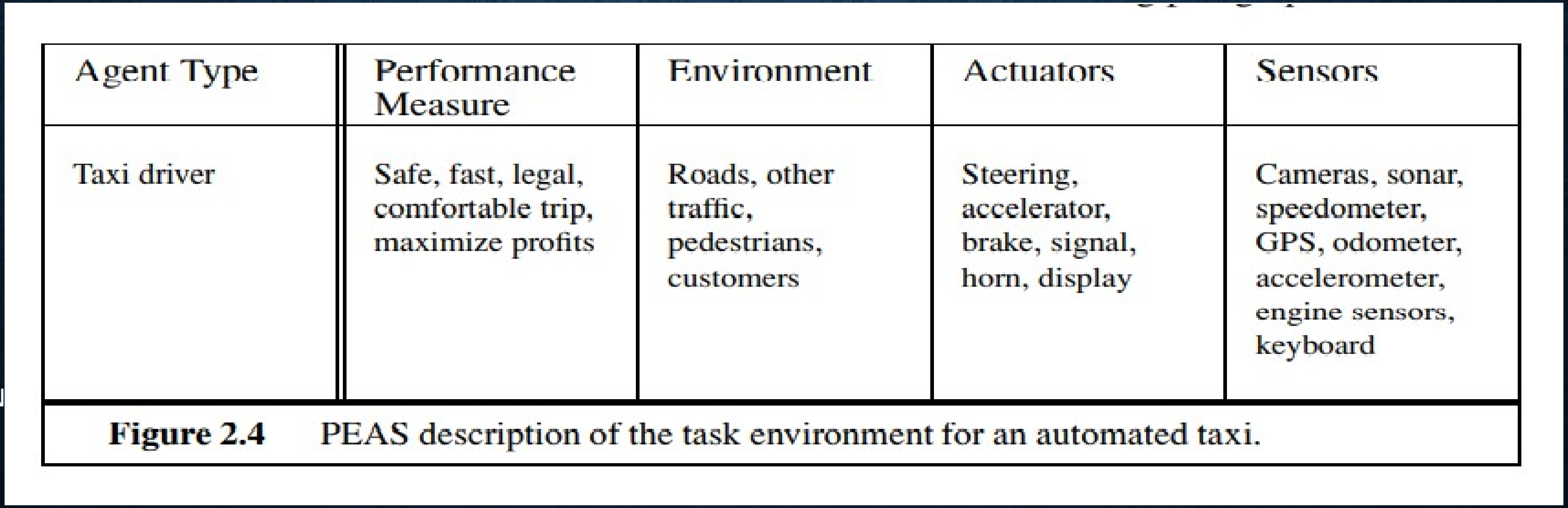
- We must understand about the task environments
- Different types of task environment and ways of defining it
- Choosing the task environment appropriate with the agent

1. SPECIFYING THE TASK ENVIRONMENT

PEAS (Performance, Environment, Actuators, Sensors)-Task Environment

Ex: Vacuum cleaner we had specified Performance measure, Environment, Actuators and Sensors)

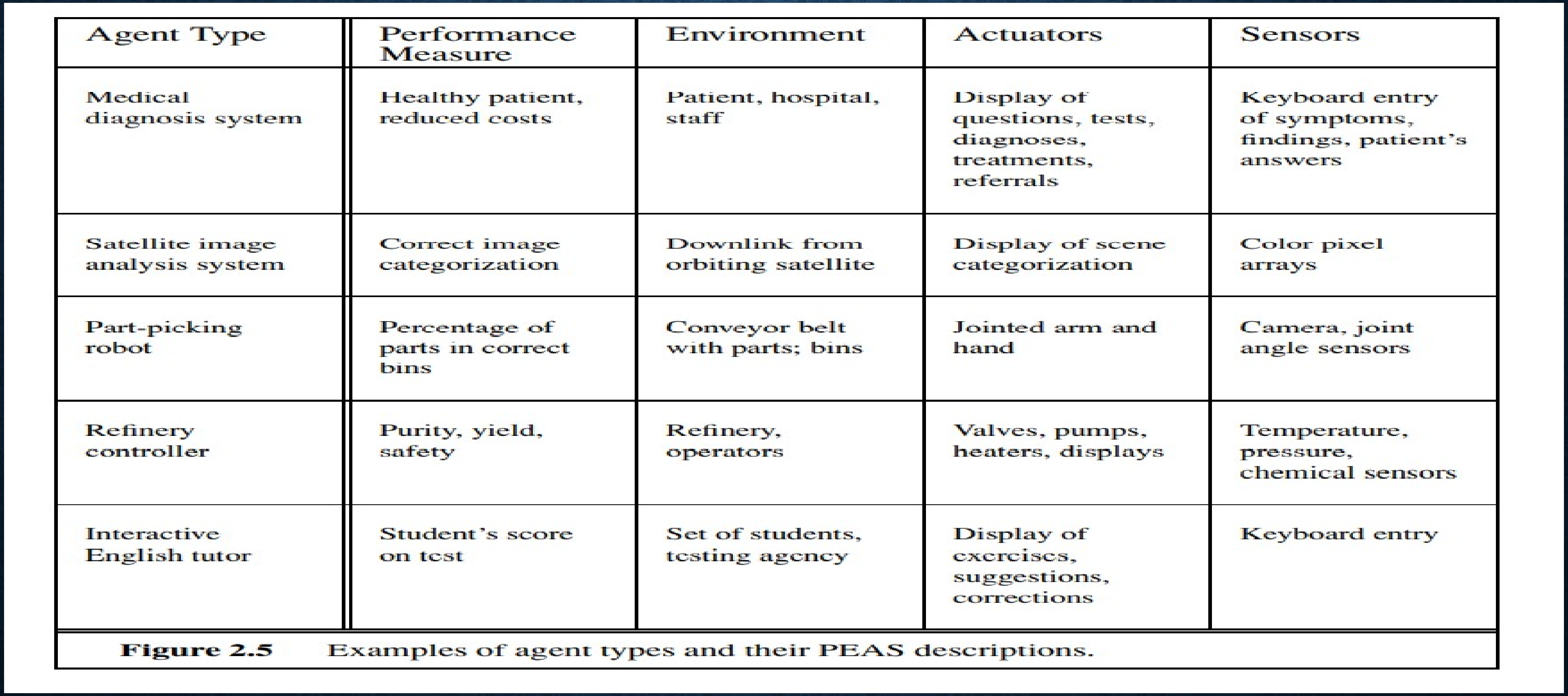
Ex 2: Automated taxi driver (still it is not completely implemented)



Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Figure 2.4 PEAS description of the task environment for an automated taxi.

SOME EXAMPLES WITH PEAS



Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Figure 2.5 Examples of agent types and their PEAS descriptions.

2.PROPERTIES OF TASK ENVIRONMENTS

Fully observable vs partially observable: If the agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable.

A task environment is effectively fully observable if the sensors detect all aspects that are relevant to the choice of the action; relevance intern depends on the performance measure.

Fully observable environments are convenient because the agent need not maintain any internal state to keep track of the world.

An Environment state is partially observable because of noisy and inaccurate sensors or because parts of the states are simply missing from the sensor data.

Ex: A Vacuum agent with only local dirt sensor cannot tell whether there is dirt in other squares, an automated taxi driver cannot see what other drivers are thinking

An agent with no sensors at all then the environment is unobservable

Single agent vs Multi agent: Consider an agent solving a crossword puzzle by itself is clearly in a single agent environment, whereas an agent playing chess is a two-agent environment.

Deterministic vs stochastic: If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic otherwise it is stochastic

Episodic vs Sequential: In an episode task environment, the agent's experience is divided into automatic episodes. In each episode the agent receives a percept and then performs a single action. Crucially, the next episode does not depend on the actions taken in previous episodes

In sequential environments, on the other hand, the current decision could affect all future decision Ex: Chess and taxi driving are sequential.

Static vs Dynamic: If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static.

Ex: Crossword puzzles are static and Taxi driving is dynamic

STRUCTURE OF AGENTS

To prepare one agent what all the things we need

Agent Behavior : It is the agent’s actions, what all the actions that an agent performs within an environment

Agent Program: How you code the agent? How you implement an agent? Which language you will use for implementation? To write an agent’s program you must know all the details about agent’s sensors. Hardware should be there then only we can code. It is one by one implementation. Ex: one percept will come and you will perform actions against it

Agent Function: It is an overall implementation. It is overall collection of percept sequence and hence we can develop the complete working of an agent.

Agent Architecture: Agent’s hardware parts

AGENT BEHAVIOUR

The action that is performed after any given sequence of percepts

We require the hardware part here is the actuator

Ex: Smoke detection system (Whenever the smoke is detected rain should come; here actuator is shower and action is rain)◊ this is what agent’s behavior in a given environment.

AGENT PROGRAM

It is basically the job of AI. Here you need to code for AI and need to inform AI that how actually your deployed sensors are going to work.

Before you write agent’s program you must know all the agent’s sensors. (Ex: you should not write some program which does not have any actuator; suppose you are writing a code on to control the leg of an agent, if suppose agent’s leg itself is not there.)

Hence we need to study hardware very well

Agent’s Function is implemented using agent program (Ex: Refrigerator function is to keep its environment cold; it is a function and is implemented using agent’s program)

AGENT PROGRAM

Input for Agent Program : Only the current percept

Output from Agent Program: Returns an action to the actuators

AGENT FUNCTION

The mapping from percepts to actions

Inputs: The Entire percept sequence. We need big database or memory to store.

It is very complicated to remember all the percepts

Agent program is the actual implementation of agent function

AGENT ARCHITECTURE

Agent program will run on some sort of computing device with physical

Agent’s sensors, actuators, any other components comes under agent architecture

NOTE: Structure of Agent is defined as

AGENT=Architecture+Program

TYPES OF AGENT PROGRAM

Simple-Reflex agents

Model-Based Reflex agent

Goal Based agents

Utility based agents

SIMPLE-REFLEX AGENTS

Reflex means immediately or Spontaneously.

In this type of agent program we have to perform actions immediately (Ex: Removing your hand immediately when you touch hot vessels)

Here the agent acts only on the basis of current perception.

It ignores the rest of the percept history

It is based on IF-THEN rules

Environment should be fully observable.

CONTD…

Ex: Tic-Tac-Toe, Vacuum cleaner

How agent will work ??

Agent will observe the environment and generate the percepts.

It will be able to understand the current situation of the environment

It will apply IF-THEN rules on the current environment and Apply Actions on it

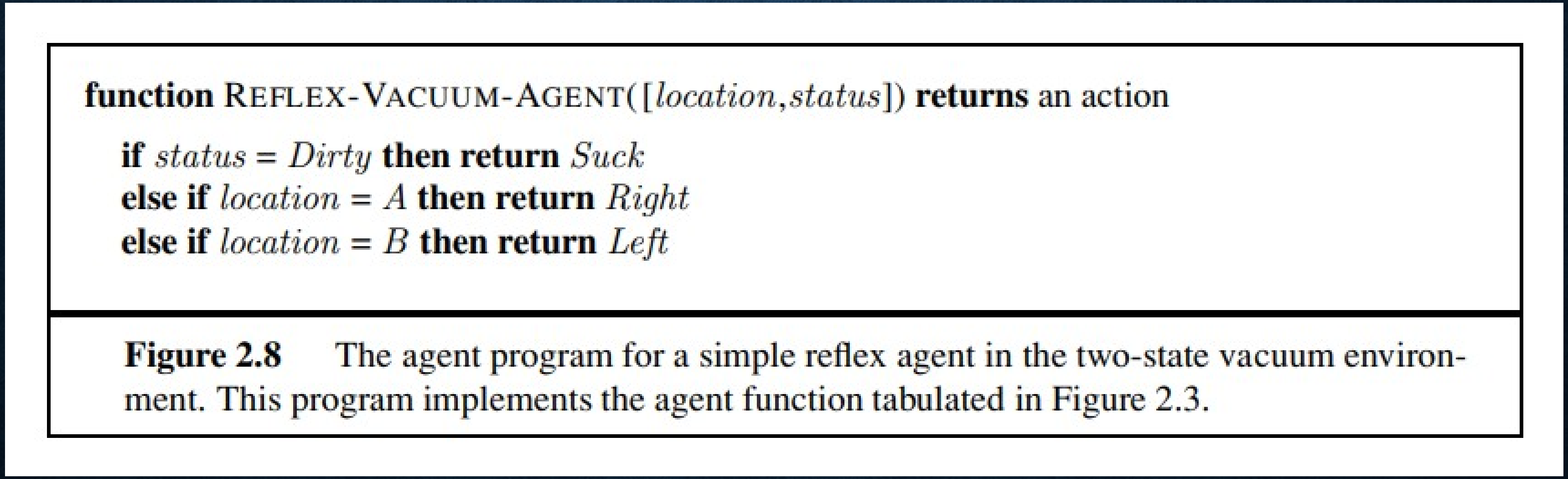
Based on the actions environment will get changed through actuators

SIMPLE REFLEX AGENT-VACUUM CLEANER

Vacuum cleaner is a simple reflex agent.

Because its decision is only based on the current location and whether that location contains dirt

Agent program for this as below



```
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
```

Figure 2.8 The agent program for a simple reflex agent in the two-state vacuum environment. This program implements the agent function tabulated in Figure 2.3.

SCHEMATIC DIAGRAM OF SIMPLE REFLEX AGENT

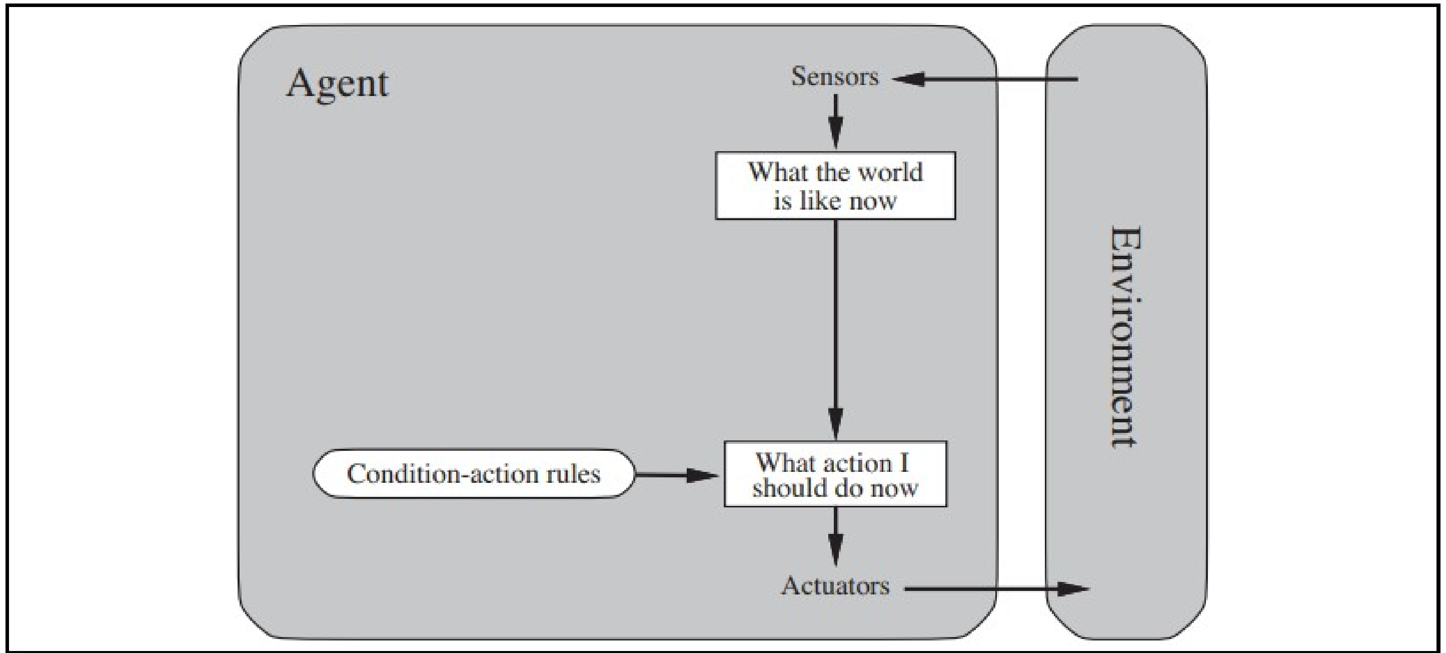
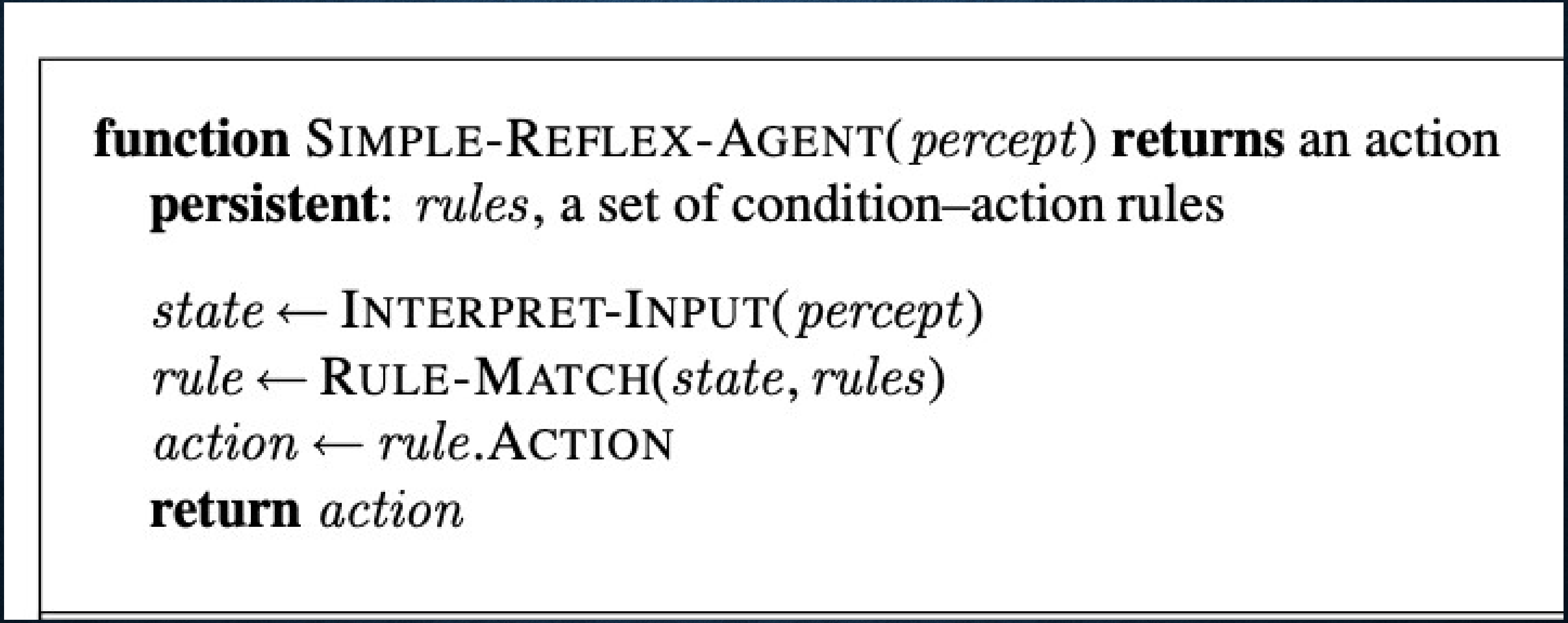


Figure 2.9 Schematic diagram of a simple reflex agent.

Slides by Disha D N

A SIMPLE REFLEX AGENT



```
function SIMPLE-REFLEX-AGENT($percept$) returns an action
    persistent: $rules$, a set of condition–action rules

    $state \leftarrow$ INTERPRET-INPUT($percept$)
    $rule \leftarrow$ RULE-MATCH($state, rules$)
    $action \leftarrow rule$.ACTION
    return $action$
```

DISADVANTAGES OF SIMPLE REFLEX AGENT

Even though it is simple to design it turns out to be a limited intelligence

It works only if the environment is fully observable

MODEL BASED REFLEX AGENT

The most effective way to handle partial observability is for agent to keep track of the part of the world it cant see now

That is, the agent should maintain some sort of internal state that depends on the percept history and there by reflects at least some of the unobserved aspects of the current state

Updating the internal state requires two kinds of knowledge to be encoded in the agent program

- First, we need some information about how the world evolves independently of the agent
- Second, how the agent’s action would affect the world

CONTD…

The knowledge about "How the world works-whether implemented in simple Boolean circuit or in complete scientific theories is called model of the world"

An agent that uses such a model is called "Model based agent"

CONTD..

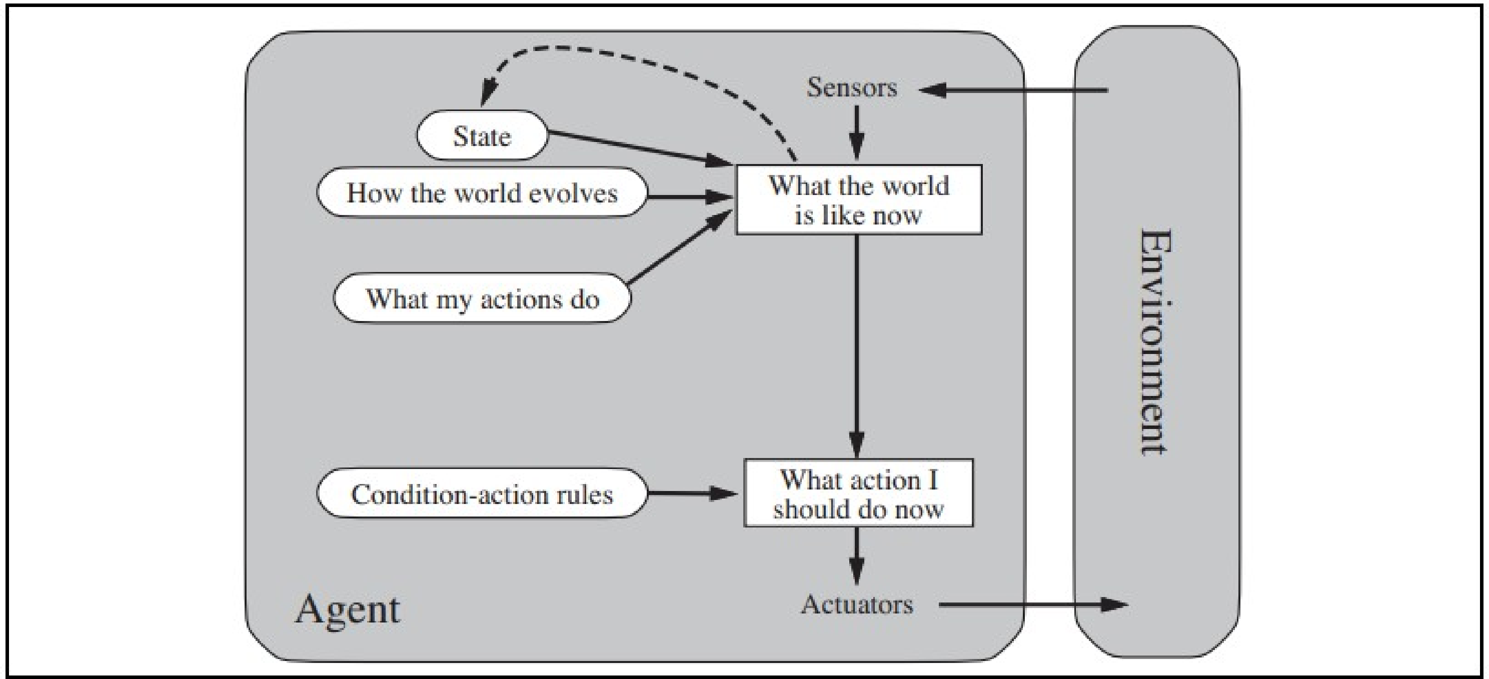
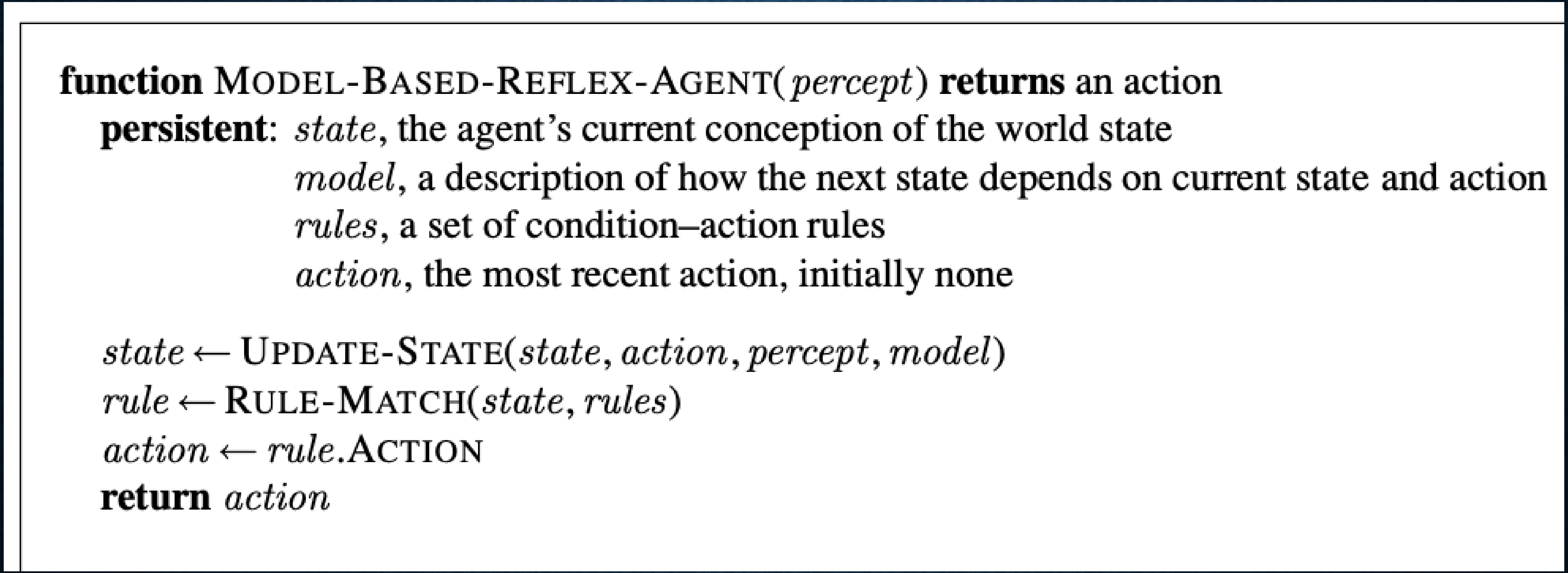


Figure 2.11 A model-based reflex agent.

CONTD..



```
**function** MODEL-BASED-REFLEX-AGENT($percept$) **returns** an action
    **persistent**: $state$, the agent's current conception of the world state
                 $model$, a description of how the next state depends on current state and action
                 $rules$, a set of condition–action rules
                 $action$, the most recent action, initially none

    $state \leftarrow$ UPDATE-STATE($state, action, percept, model$)
    $rule \leftarrow$ RULE-MATCH($state, rules$)
    $action \leftarrow rule$.ACTION
    **return** $action$
```

GOAL BASED AGENT

Knowing something about the current state of the environment is not always enough to decide what to do (Ex: at a road junction, taxi can turn left, right or go straight on)

The current decision depends on where the taxi is trying to get to. That is as well as a current state description the agent needs some sort of goal information that describes situations that are desirable.

Agent Program can combine this with the model to choose actions that achieve the goal

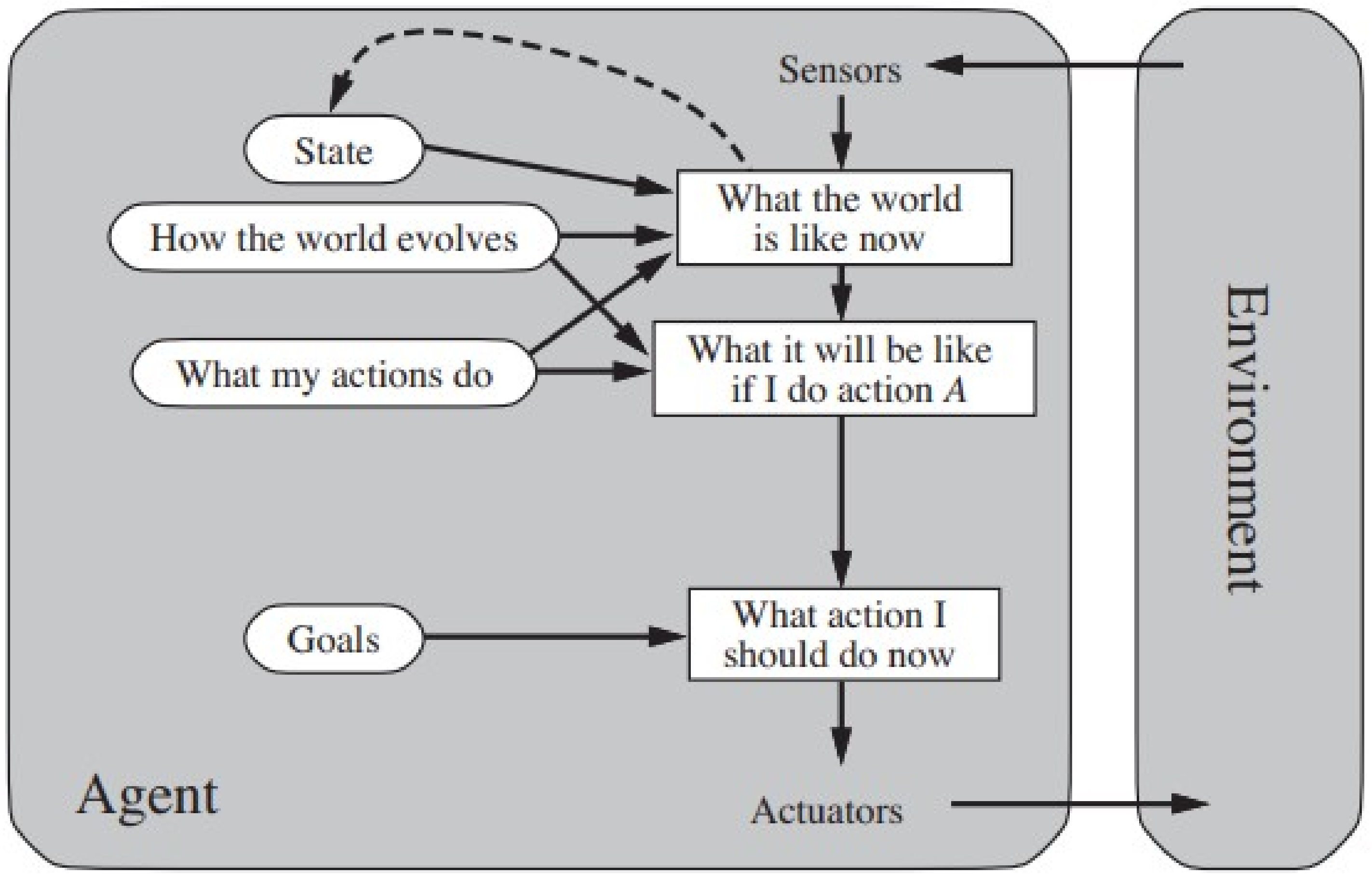

Sensors
State
What the world is like now
How the world evolves
What my actions do
What it will be like if I do action *A*
Goals
What action I should do now
Agent
Actuators
Environment

Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

UTILITY BASED AGNET

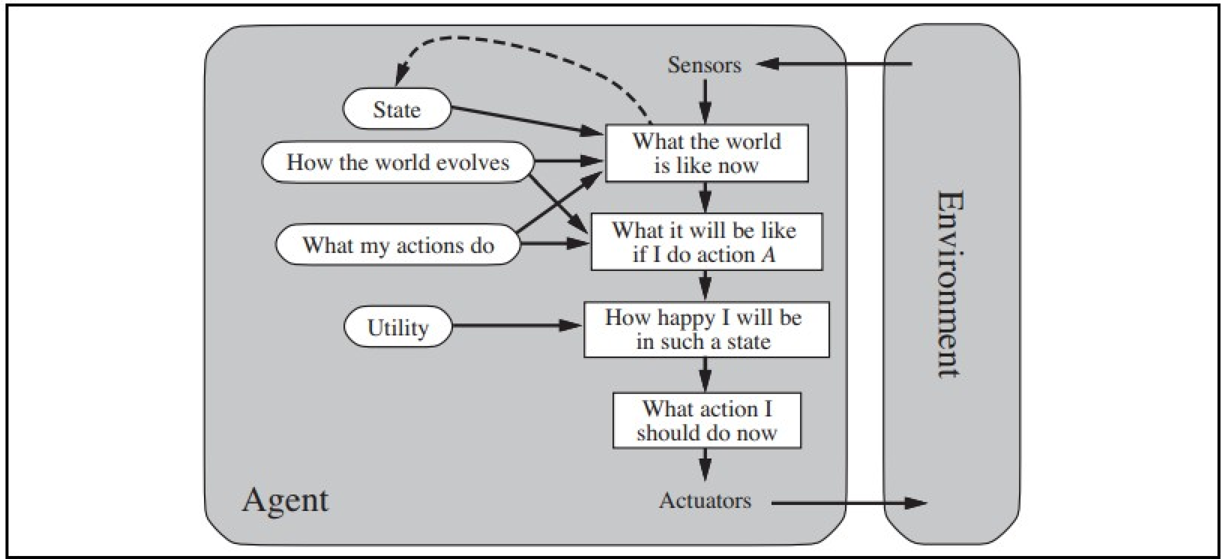
Goals alone are not enough to generate high-quality behaviour in most environments.

Ex: Many action sequences will get the taxi to its destination(there by achieving goal) but some are quicker, safer, more reliable or cheaper than others. Goals just provide a crude distinction between “happy” and “unhappy” states

More scientific way of describing happy or unhappy states is by the term utility based

Here, utility function is essentially an internalization of the performance measure. If the internal utility function and external performance measure are in agreement, then an agent chooses and action to maximize its utility.

CONTD..



Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action *A*
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

Figure 2.14 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

UTILITY BASED AGNET

a utility-based agent has many advantages in terms of flexibility and learning

when there are conflicting goals, only some of which can be achieved (for example, speed and safety), the utility function specifies the appropriate tradeoff.

there are several goals that the agent can aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighed against the importance of the goals.

there are several goals that the agent can aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighed against the importance of the goals.

SOLVING PROBLEMS BY SEARCHING

CHAPTER 3

Slides by Disha D N

TOPICS TO LEARN

Problem solving Agents

Well defined problem and solution

Formulating Problem : Examples

Searching for solutions

Uninformed search strategies

Informed (Heuristic Search Strategies)

Heuristic Functions

PROBLEM SOLVING AGENT

It is one of the type among Goal Based Agent

Goal formulation, based on the current situation and agent's performance measure, is the first step in problem solving

Agent's task is to find out and decide how to act now and in the future, so that it reaches a goal state, it can be done by problem formulation

Problem formulation is the process of deciding what actions and states to consider, given a goal.

Goals can be reached quickly and efficiently if agent knows about the environment.

CONTD..

We assume that the environment is always observable, known, discrete and deterministic so the agent always knows the current state.

Under these assumptions, the solution to any problem is a fixed sequence of actions

The process of looking for a sequence of actions that reaches the goal is called search.

A search algorithm takes a problem as input and returns a solution in the form of an action sequence. Once the solution is found the actions it recommends can be carried out and this is called as execution phase.

After formulating a goal and a problem to solve, the agent calls a search procedure to solve it

WELL DEFINED PROBLEMS AND SOLUTIONS

A Problem can be defined formally by five components

The Initial State that the agent starts in.

Actions: Description of possible actions available to the agent. Given a particular state s, ACTIONS(s) returns the set of actions that can be executed in s. We say that each of these actions is applicable in s.

Transition Model: The description of what each action does; the formal name for this is transition model, specified by a function RESULT(s,a) that returns the state that results from doing action a in state s

We also use the term successor to refer to any state reachable from a given state by a single action

Together, initial state, actions and transition model implicitly define the state space of the problem-set of all states reachable from the initial state by any sequence of actions

The state space can be represented by a network or a graph in which the nodes are states and the links between nodes are actions

Goal test which determines whether the given state is a goal state (Ex: in chess, the goal is to reach a state called "checkmate"

Path cost function assigns a numeric cost to each path. The problem solving agent chooses a cost function that reflects its own performance measure.

Solution quality is measured by path cost function, an optimal solution has the lowest path cost among all the solution

EXAMPLE PROBLEMS

Here we understand two different kinds of problem, Toy problem and Real world problem.

Toy problem is intended to illustrate or exercise various problem solving methods. It can be given concise, exact description and hence is usable by different researcher to compare the performance of algorithms

A real world problem is one whose solutions people actually care about.

TOY PROBLEMS

The first example we examine here is the Vacuum World and it can be formulated as problem as below

States: The state is determined by both the agent location and the dirt locations. The agent is in one of the two locations, each of might or might not contain dirt. Thus there are 2*2^2=8 possible states. A larger environment with n locations has n*2^n states.

Initial State: Any state can be designated as the initial state.

Actions: In this simple environment, each state has just three actions: Left, Right, and Suck. Larger environments may have Up and Down as well

CONTD..

Transition model: The actions they have their expected effects, except that moving Left in leftmost square, moving Right in rightmost square and Suck in clean square have no effect. The complete state space is shown in the figure 3.3

Goal Test: This checks whether all the squares are clean

Path Text: Each step costs 1, hence the path cost is the number of steps in the path.

CONTD..

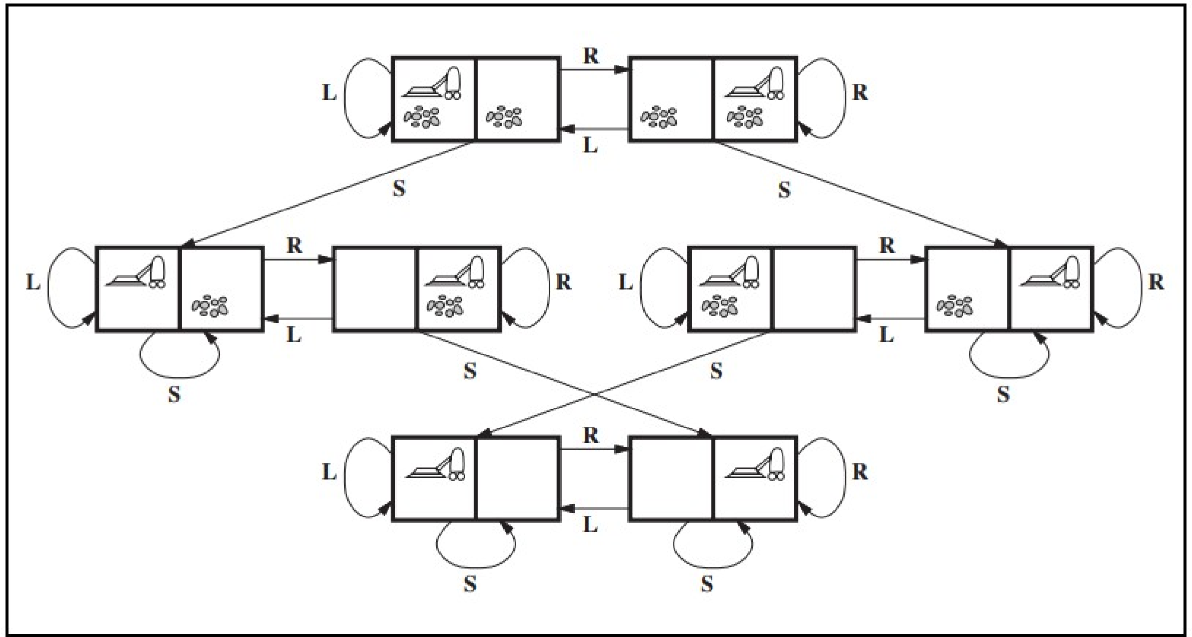


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

8-QUEEN PROBLEM

Goal is to place eight queens on a chessboard such that no queens attacks any other.

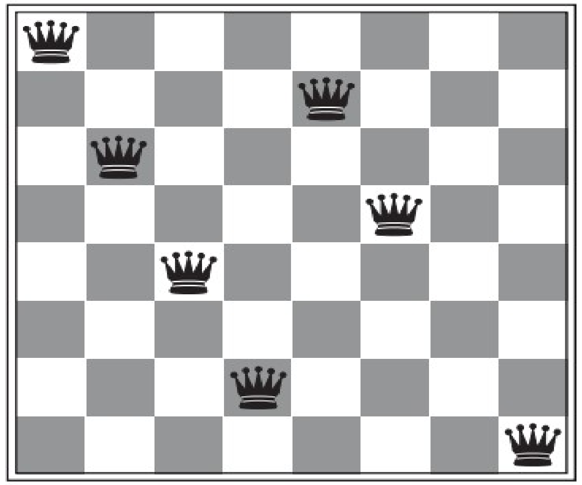


Figure 3.5 Almost a solution to the 8-queens problem. (Solution is left as an exercise.)

CONTD..

For incremental formulation, problem formulation can be defined as

State: Any arrangement of 0 to 8 queens on the board is a state.

Initial State: No queens on the board

Actions: Add a queen to any empty square

Transition model: Returns the board with a queen added to the specified square.

Goal Test: 8 queens on the board, none attacked.

REAL WORLD PROBLEMS-ROUTE FINDING ALGORITHM

Consider the airline travel problems that must be solved by a travel planning website.

States: Each state obviously includes location and the current time. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their fare buses, an their status as domestic or international, the state must record extra information about these “historical” aspects.

Actions: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.

Transition model: The state resulting from taking a flight will have the flight’s destination as the current location and the flight’s arrival time as the current time.

Goal Test: Are we at the final destination specified by the user?

Path cost: This depends on monetary cost, waiting time, flight time, customers and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage and so on.

SEARCHING FOR SOLUTIONS

Once problems are formulated we need to solve them. A solution is an action sequence, so search algorithms work by considering various possible action sequences.

The possible action sequences starting at the initial state from a search tree with the initial state at the root; the branches are actions and the nodes correspond to states in the state space of the problem.

Figure 3.6 shows the first few steps in growing the search tree for finding a route

The route node of the tree corresponds to initial state. The first step is to test whether this is a goal state. Then we need to consider taking various actions.

We do this by expanding the current state; that is, each applying each legal actions to the current state, there by generating a new set of states.

In this case, we add three branches from parent node, leading to three child nodes.

Node with no child nodes are called as leaf node.

The set of all leaf nodes available for expansion at any given point is called the frontier.

The process of expanding nodes on the frontier continues until either solution is found or no more states to expand

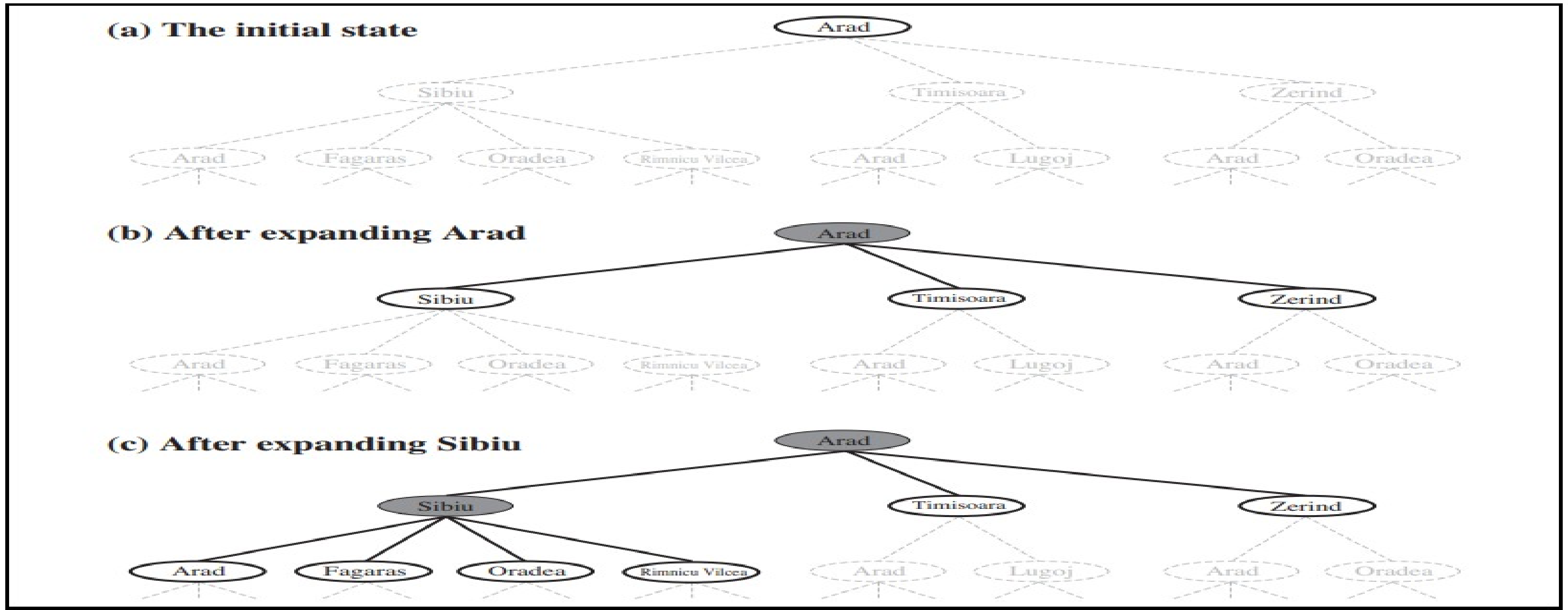


Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

INFRASTRUCTURE FOR SEARCH ALGORITHMS

Search algorithms require a data structure to keep track of the search tree that is being constructed. For each node n of the tree, we have a structure that contains four components:

n.State: The state in the state space to which the node corresponds;

n.Parent: The node in the search tree that generated this node;

n.Actions: The action that was applied to the parent to generate the node;

n.Path_cost: the cost, traditionally denoted by g(n), of the path from initial state to the node, as indicated by the parent pointers.

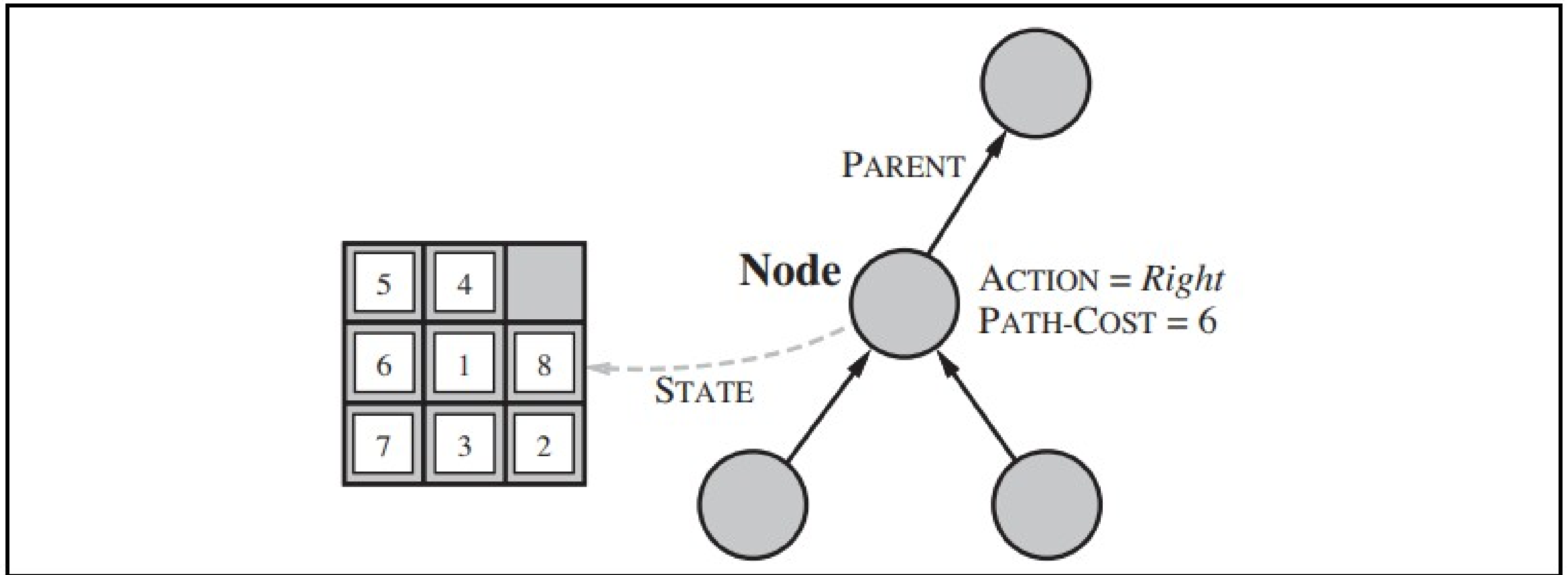


Figure 3.10 Nodes are the data structures from which the search tree is constructed. Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent.

MEASURING PROBLEM SOLVING PERFORMANCE

Here we understand, how to evaluate algorithm performance

Completeness: Is the algorithm guaranteed to find a solution when there is one?

Optimality: Does the strategy find the optimal solution?

Time Complexity: How long does it take to find a solution

Space Complexity: How much memory is needed to perform the search?

UNINFORMED SEARCH STRATEGIES

It is also called as blind search.

Here, the search strategies have no additional information about states beyond that provided in the problem definition.

All they can do is generate successors and distinguish a goal state from a non goal state

All search strategies are distinguished by the order in which nodes are expanded.

BREADTH-FIRST SEARCH

It is a simple strategy in which root node is expanded first, then all the successors of the root node are expanded next, then their successors and so on.

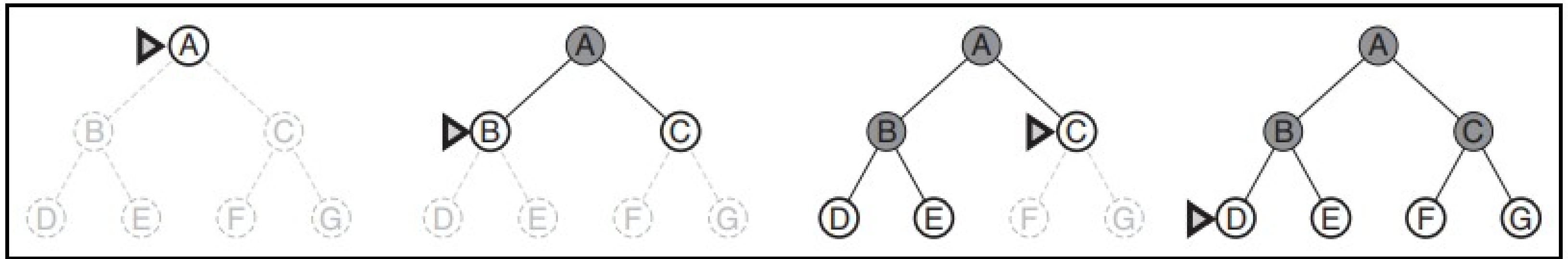
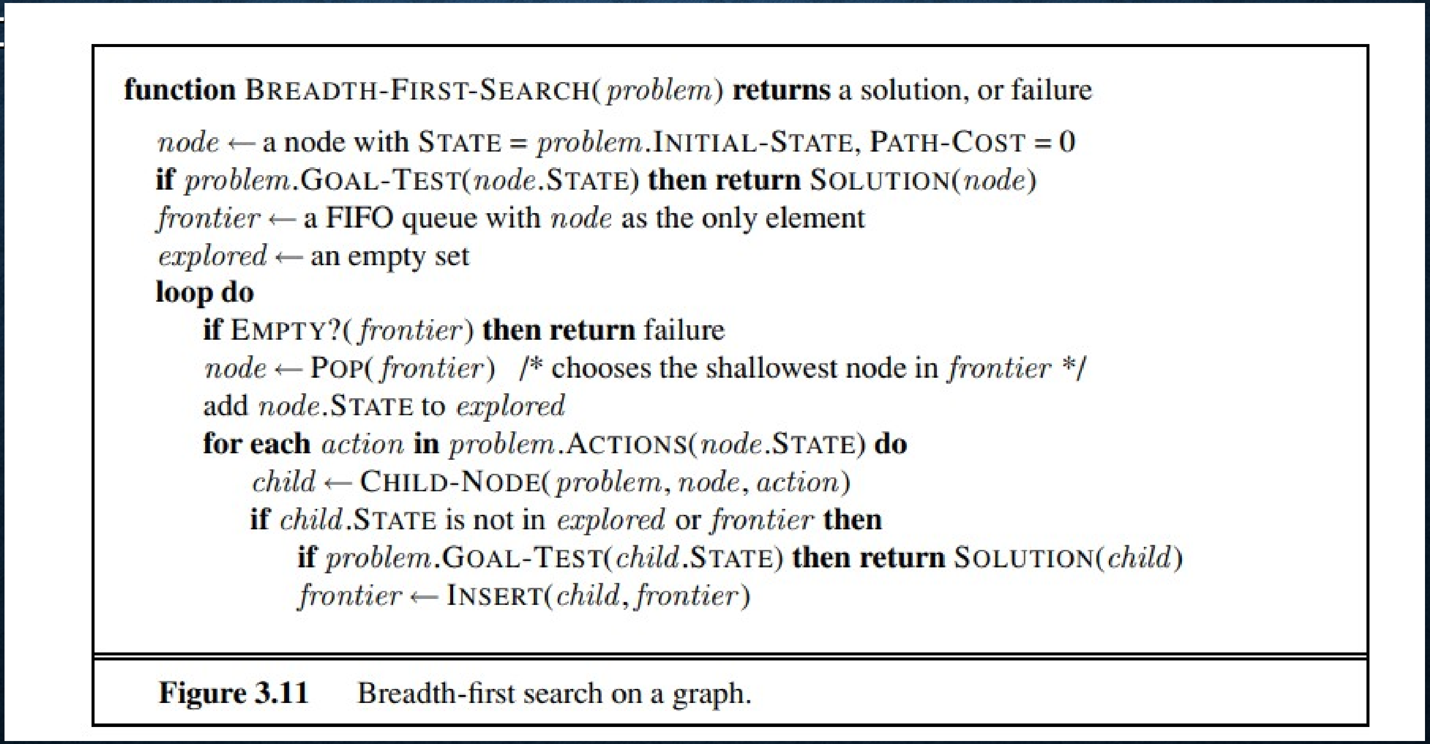


Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

CONTD..

BFS can be implemented using FIFO queue for the frontier. Thus new nodes go to the back of the queue, and old queues which are shallower than the new nodes, get



```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

Slides by Disha D N

UNIFORMED COST SEARCH

When all step costs are equal, breadth first search is optimal because it always expands the shallowest unexpanded node.

By a simple extension, we can find an algorithm that is optimal with any step-cost function. Instead of expanding the shallowest node, uniform-cost search expands the node n with the lowest path cost g(n). This is done by storing the frontier as a priority queue ordered by g.

In addition to the ordering of the queue by path cost, there are two other significant differences from breadth first search. The first is that the goal test is applied to a node when it is selected for expansion rather than when it is first generated.

The second difference is that a test is added in case a better path is found to a node currently on the frontier.

DEPTH FIRST SEARCH

DFS always expands the deepest node in the frontier of the search tree

The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors.

As those nodes are expanded, they are dropped from the frontier, so then the search “backs up” to the next deepest node that still has unexplored successors.

DFS is a LIFO queue. It means, a most recently generated node is chosen for expansion.

DFS is explained as in figure

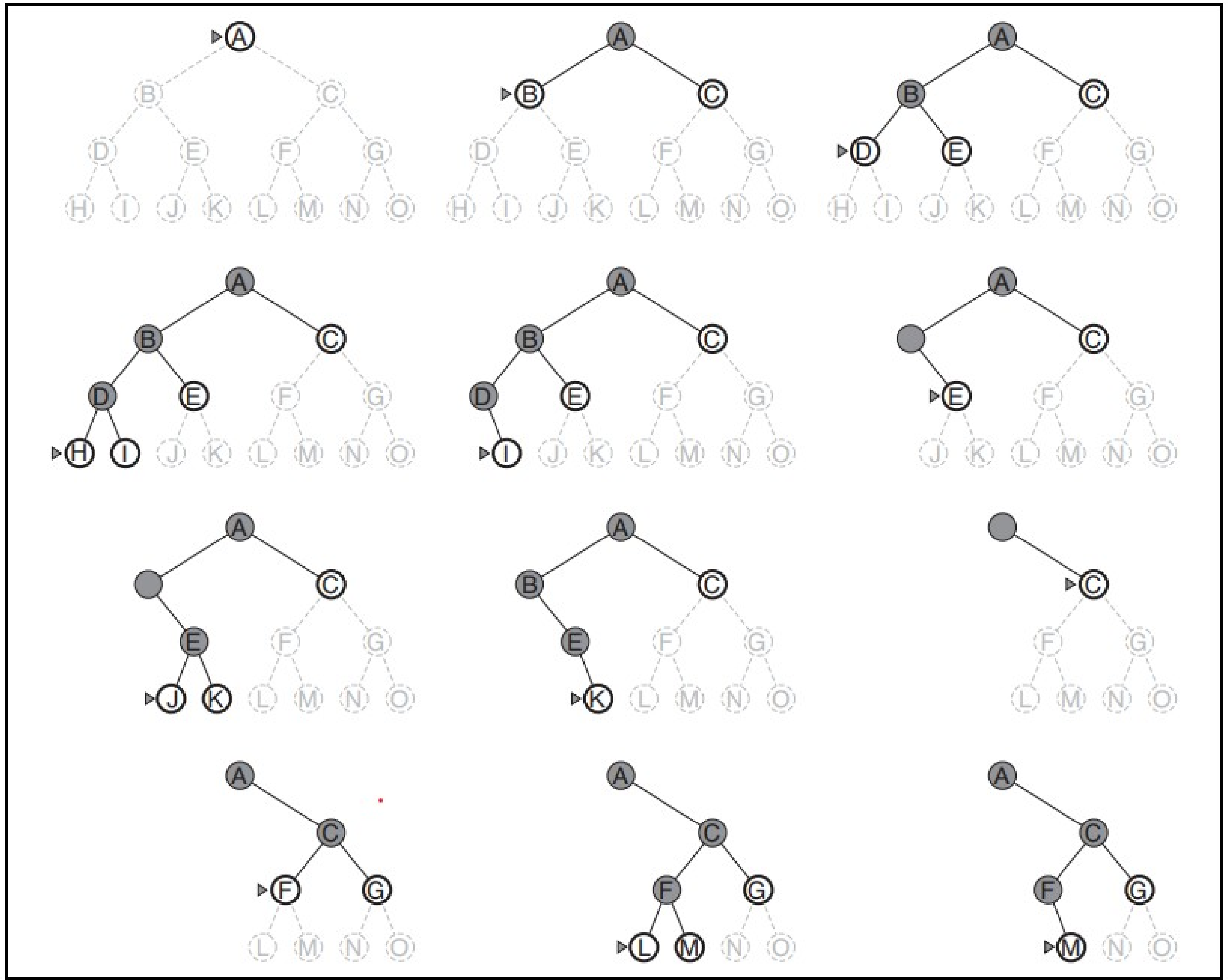


Figure 3.16 Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

Slides by Disha D N

INFORMED(HEURISTIC) SEARCH STRATEGIES

Informed search strategy uses problem specific knowledge beyond the definition of the problem itself-information about the goal is known in prior

It finds the optimal solution to reach the goal state using heuristic function

Informed search can be defined as a search which tries to reduce the amount of search that must be done by making intelligent choices among the nodes that are selected for the expansion.

Here we use heuristic function to find heuristic value which will be combined with path cost function to find the optimal solution

Example: 1 Best fit search algorithm

Example 2: A* search algorithm

GREEDY BEST FIRST SEARCH

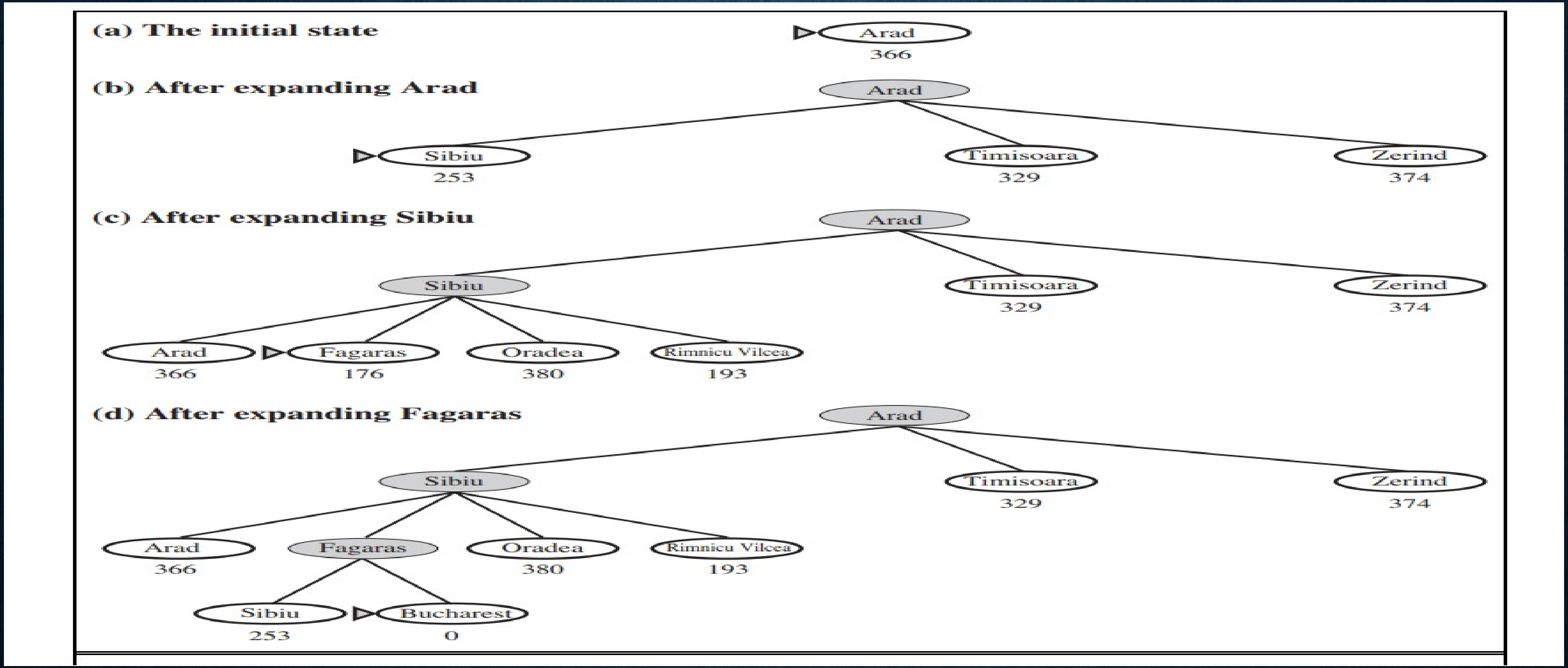
Greedy best first search tries to expand the node that is closest to the goal. It leads to the solution quickly.

It evaluates the node by using just the heuristic function; that is f(n)=h(n)

If it is a route finding problems, we use hSLD. (heuristic straight line distance)

Value of hSLD cannot be computed from the problem description itself, it takes a certain amount of experience to know that hSLD is correlated with actual road distances

CONTD..



Slides by Disha D N

CONTD..

The first node to be expanded from Arad will be Sibiu because it is closer to Bucharest than either Zerind or Timisoara.

The next node to be expanded will be Fagaras because it is closest. Fagaras in turn generates Bucharest, which is the goal.

For this particular problem, greedy best-first search using hSLD finds a solution without ever expanding a node that is not on the solution path

hence, its search cost is minimal.

It is not optimal, however: the path via Sibiu and Fagaras to Bucharest is 32 kilometers longer than the path through Rimnicu Vilcea and Pitesti.

This shows why the algorithm is called “greedy”—at each step it tries to get as close to the goal as it can

VISUALIZATION OF GREEDY ALGORITHM

Click here

Slides by Disha D N

KNOWLEDGE-ENHANCING VIDEO

Click here

Slides by Disha D N

A* SEARCH: MINIMIZING THE TOTAL ESTIMATED SOLUTION COST

The most widely known form of best first search is called A* search

It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal

$f(n)=g(n)+h(n)$

Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal.

f(n)= estimated cost of the cheapest solution through n

Hence A* search is both complete and optimal.

CONDITIONS FOR OPTIMALITY: ADMISSIBILITY AND CONSISTENCY

The first condition we require for optimality is that h(n) be an admissible heuristic.

An admissible heuristic is one that never overestimates the cost to reach the goal.

Because g(n) is the actual cost to reach n along the current path, and f(n)=g(n)+h(n).

Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than actually is.

An obvious example of an admissible heuristic is the straight-line distance hSLD. It is admissible because the shortest path between any two points

Second, slightly stronger condition called consistency is required only for applications of A* to graph search.

A heuristic h(n) is consistent if, for every node n and every successor n of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n plus the estimated cost of reaching the goal from n : $h(n) \leq c(n, a, n) + h(n)$.