

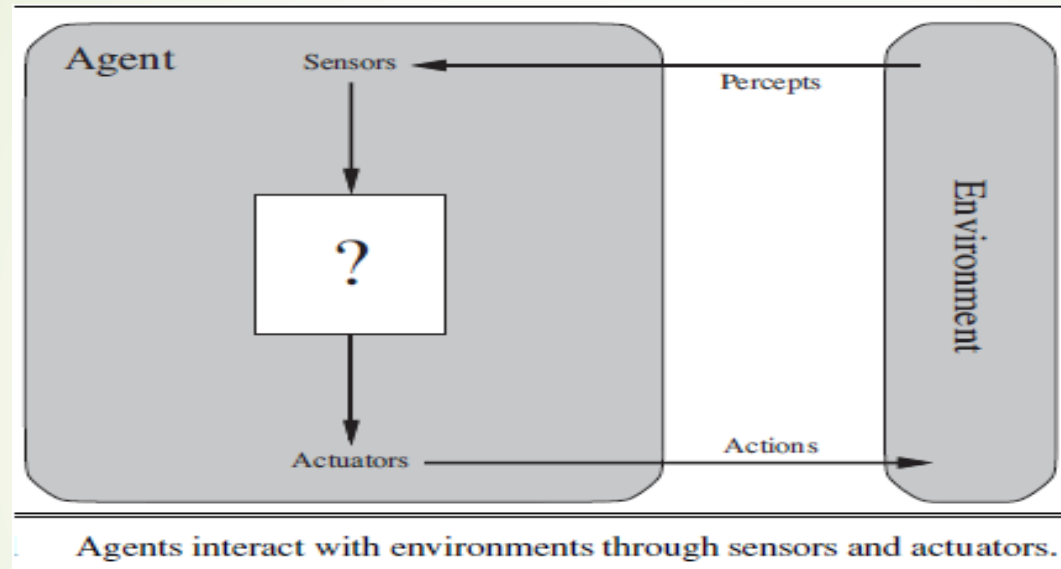


CHAPTER 2: Intelligent System

Prepared By:
Dipanita Saha
Assistant Professor
Institute of Information Technology(IIT)
Noakhali Science and Technology University

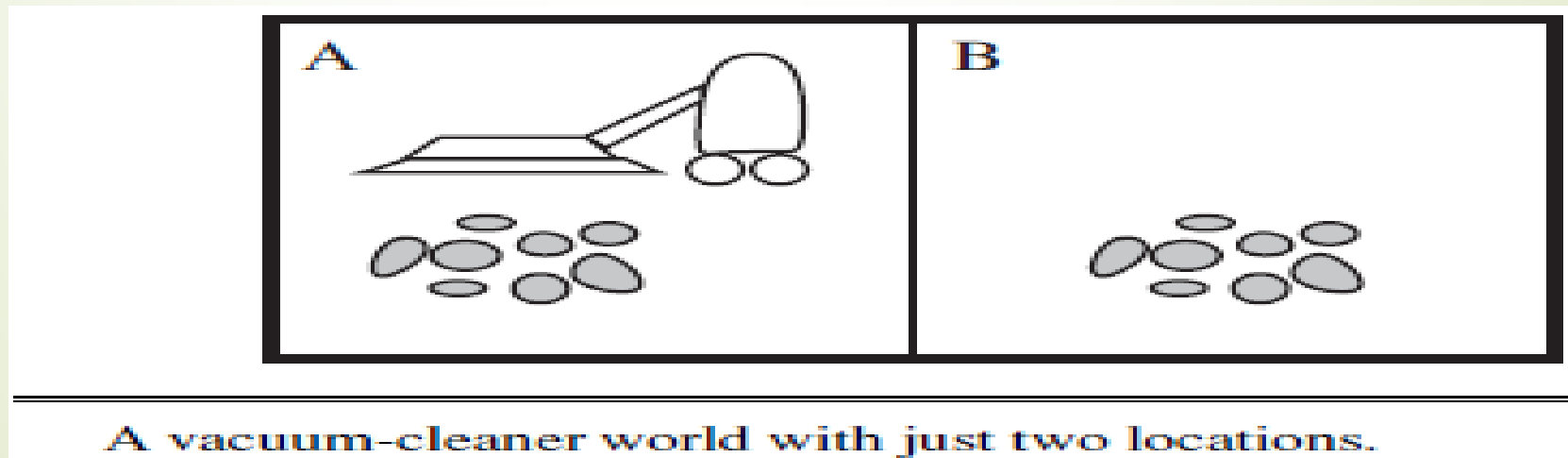
Agent and Environments

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



- A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators.
- A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.
- 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.

- **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.
- Mathematically speaking, an agent's behavior is described by the **agent function** that maps any given percept sequence to an action.
- the agent function for an artificial agent will be implemented by an **agent program**.
- It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running within some physical system.
- The vacuum-cleaner world shown in Figure 2.2. This particular world has just 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 this agent function is shown in Figure and an agent program that implements it appears in Figure.

Percept sequence	Action
<i>[A, Clean]</i>	<i>Right</i>
<i>[A, Dirty]</i>	<i>Suck</i>
<i>[B, Clean]</i>	<i>Left</i>
<i>[B, Dirty]</i>	<i>Suck</i>
<i>[A, Clean], [A, Clean]</i>	<i>Right</i>
<i>[A, Clean], [A, Dirty]</i>	<i>Suck</i>
<i>⋮</i>	<i>⋮</i>
<i>[A, Clean], [A, Clean], [A, Clean]</i>	<i>Right</i>
<i>[A, Clean], [A, Clean], [A, Dirty]</i>	<i>Suck</i>
<i>⋮</i>	<i>⋮</i>

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world


GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Rationality

- What is rational at any given time depends on four things:
 - The performance measure that defines the criterion of success.
 - The agent's prior knowledge of the environment.
 - The actions that the agent can perform.
 - The agent's percept sequence to date

➤ Definition of a 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.
- Consider the simple vacuum-cleaner agent that cleans a square if it is dirty and moves to the other square if not; Is this a rational agent? That depends!
- ❑ The performance measure awards one point for each clean square at each time step, over a “lifetime” of 1000 time steps.

- 
- ❑ The “geography” of the environment is known *a priori* but the dirt distribution and the initial location of the agent are not. Clean squares stay clean and sucking cleans the current square. The Left and Right actions move the agent left and right except when this would take the agent outside the environment, in which case the agent remains where it is.
 - ❑ The only available actions are Left , Right, and Suck.
 - ❑ The agent correctly perceives its location and whether that location contains dirt.
 - One can see easily that the same agent would be **irrational** under different circumstances.
 - For example, once all the dirt is cleaned up, the agent will oscillate needlessly back and forth;
 - if the performance measure includes a penalty of one point for each movement left or right, the agent will fare poorly.
 - A better agent for this case would do nothing once it is sure that all the squares are clean. If clean squares can become dirty again, the agent should occasionally check and re-clean them if needed.
 - If the geography of the environment is unknown, the agent will need to explore it rather than stick to squares A and B.



Omniscience, learning, and autonomy

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

- Consider the following example: I am walking along the Champs Elysées 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, 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."
- This example shows that rationality is not the same as perfection. Rationality maximizes *expected* performance, while perfection maximizes *actual* performance.
- Our definition of rationality does not require omniscience, then, because the rational choice depends only on the percept sequence *to date*.
- We must also ensure that we haven't inadvertently allowed the agent to engage in decidedly under intelligent activities.

- For example, if an agent does not look both ways before crossing a busy road, then its percept sequence will not tell it that there is a large truck approaching at high speed.
- Does our definition of rationality say that it's now OK to cross the road?
- Far from it! First, it would not be rational to cross the road given this uninformative percept sequence: the risk of accident from crossing without looking is too great.
- Second, a rational agent should choose the “looking” action before stepping into the street, because looking helps maximize the expected performance.
- Doing actions *in order to modify future percepts*—sometimes called **information gathering**— is an important part of rationality.
- A second example of information gathering is provided by the **exploration** that must be undertaken by a vacuum-cleaning agent in an initially unknown environment.

- The agent's initial configuration could reflect some prior knowledge of the environment, but as the agent gains experience this may be modified and augmented.
- There are extreme cases in which the environment is completely known *a priori*. In such cases, the agent need not perceive or learn; it simply acts correctly.
- Of course, such agents are fragile.
- Consider the lowly dung beetle. After digging its nest and laying its eggs, it fetches a ball of dung from a nearby heap to plug the entrance. If the ball of dung is removed from its grasp *en route*, the beetle continues its task and pantomimes plugging
- The nest with the nonexistent dung ball, never noticing that it is missing. Evolution has built an assumption into the beetle's behavior, and when it is violated, unsuccessful behavior results.
- Slightly more intelligent is the sphex wasp. The female sphex will dig a burrow, go out and sting a caterpillar and drag it to the burrow, enter the burrow again to check all is well, drag the caterpillar inside, and lay its eggs.
- The caterpillar serves as a food source when the eggs hatch. So far so good, but if an entomologist moves the caterpillar a few inches away while the sphex is doing the check, it will revert to the "drag" step of its plan and will continue the plan without modification, even after dozens of caterpillar-moving interventions.
- The sphex is unable to learn that its innate plan is failing, and thus will not change it.


- 
- 
- To the extent that an agent relies on the prior knowledge of its designer rather than on its own percepts, we say that the agent lacks **autonomy**.
 - A rational agent should be autonomous—it should learn what it can to compensate for partial or incorrect prior knowledge.
 - For example, a vacuum-cleaning agent that learns to foresee where and when additional dirt will appear will do better than one that does not.
 - As a practical matter, one seldom requires complete autonomy from the start: when the agent has had little or no experience, it would have to act randomly unless the designer gave some assistance.
 - So, just as evolution provides animals with enough built-in reflexes to survive long enough to learn for themselves, it would be reasonable to provide an artificial intelligent agent with some initial knowledge as well as an ability to learn.
 - After sufficient experience of its environment, the behavior of a rational agent can become effectively *independent* of its prior knowledge.

The nature of Environment

1. Specifying the task environment

- **PEAS** (Performance, Environment, Actuators, Sensors)
- In designing an agent, the first step must always be to specify the task environment as fully as possible.
- An automated taxi driver. We should point out, before the reader becomes alarmed, that a fully automated taxi is currently somewhat beyond the capabilities of existing technology.
- The full driving task is extremely *open-ended*. There is no limit to the novel combinations of circumstances.
- Figure summarizes the PEAS description for the taxi's task environment.



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







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


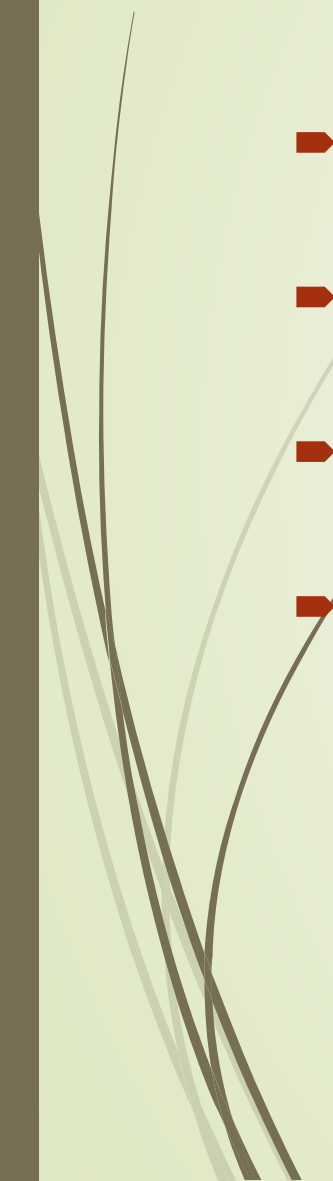
2. Properties of task environments

- **Fully observable vs. partially observable:** If an 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 action, 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 might be **partially observable** because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data.
- for example, a vacuum agent with only a local dirt sensor cannot tell whether there is dirt in other squares, and an automated taxi cannot see what other drivers are thinking. If the agent has no sensors at all then the environment is **unobservable**.

- 
- 
- **Single agent vs. multiagent:** For example, an agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a two agent environment.
 - First, we have described how an entity *may* be viewed as an agent, but we have not explained which entities *must* be viewed as agents.
 - Does an agent A (the taxi driver for example) have to treat an object B (another vehicle) as an agent, or can it be treated merely as an object behaving according to the laws of physics,
 - The key distinction is whether B's behavior is best described as maximizing a performance measure whose value depends on agent A's behavior. For example, in chess, the opponent entity B is trying to maximize its performance measure, which, by the rules of chess, minimizes agent A's performance measure.
 - Thus, chess is a **competitive** multiagent environment.
 - In the taxi-driving environment, avoiding collisions maximizes the performance measure of all agents, so it is a partially **cooperative** multiagent environment.
 - It is also partially competitive because, for example, only one car can occupy a parking space.
 - The agent-design problems in multiagent environments are often quite different from those in single-agent environments;

- 
- 
- **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.
 - In principle, an agent need not worry about uncertainty in a fully observable, deterministic environment.
 - If the environment is partially observable, however, then it could *appear* to be stochastic.
 - Most real situations are so complex that it is impossible to keep track of all the unobserved aspects; for practical purposes, they must be treated as stochastic.
 - Taxi driving is clearly stochastic in this sense, because one can never predict the behavior of traffic exactly; moreover, one's tires blow out and one's engine seizes up without warning.
 - The vacuum world as we described it is deterministic, but variations can include stochastic elements such as randomly appearing dirt and an unreliable suction mechanism.

- 
- 
- **Episodic vs. sequential:** In an episodic task environment, the agent's experience is divided into atomic 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.
 - Many classification tasks are episodic.
 - For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions; moreover, the current decision doesn't affect whether the next part is defective.
 - In sequential environments, on the other hand, the current decision could affect all future decisions.
 - Chess and taxi driving are sequential: in both cases, short-term actions can have long-term consequences.
 - Episodic environments are much simpler than sequential environments because the agent does not need to think ahead.

- 
- 
- **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.
 - Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time.
 - Dynamic environments, are continuously asking the agent what it wants to do; if it hasn't decided yet, that counts as deciding to do nothing.
 - If the environment itself does not change with the passage of time but the agent's performance score does, then we say the environment is **semidynamic**.
 - Taxi driving is clearly dynamic: the other cars and the taxi itself keep moving while the driving algorithm dithers about what to do next.
 - Chess, when played with a clock, is semidynamic. Crossword puzzles are static.

- **Discrete vs. continuous:** The discrete/continuous distinction applies to the *state* of the environment, to the way *time* is handled, and to the *percepts* and *actions* of the agent.
- For example, the chess environment has a finite number of distinct states.
- Chess also has a discrete set of percepts and actions.
- Taxi driving is a continuous-state and continuous-time problem: the speed and location of the taxi and of the other vehicles sweep through a range of continuous values and do so smoothly over time.
- Taxi-driving actions are also continuous (steering angles, etc.).
- **Known vs. unknown:** In a known environment, the outcomes for all actions are given.
- if the environment is unknown, the agent will have to learn how it works in order to make good decisions.
- the distinction between known and unknown environments is not the same as the one between fully and partially observable environments.
- It is quite possible for a *known* environment to be *partially* observable—for example, in solitaire card games, I know the rules but am still unable to see the cards that have not yet been turned over.
- Conversely, an *unknown* environment can be *fully* observable—in a new video game, the screen may show the entire game state but I still don't know what the buttons do until I try them.

THE STRUCTURE OF AGENTS

- The job of AI is to design an **agent program** that implements the agent function, the mapping from percepts to actions.
- We assume this program will run on some sort of computing device with physical sensors and actuators—we call this the **architecture**:
- $agent = architecture + program$.
- ❖ **Agent programs**
 - four basic kinds of agent programs that embody the principles underlying almost all intelligent systems:
 1. Simple reflex agents;
 2. Model-based reflex agents;
 3. Goal-based agents; and
 4. Utility-based agents.

Simple reflex agents

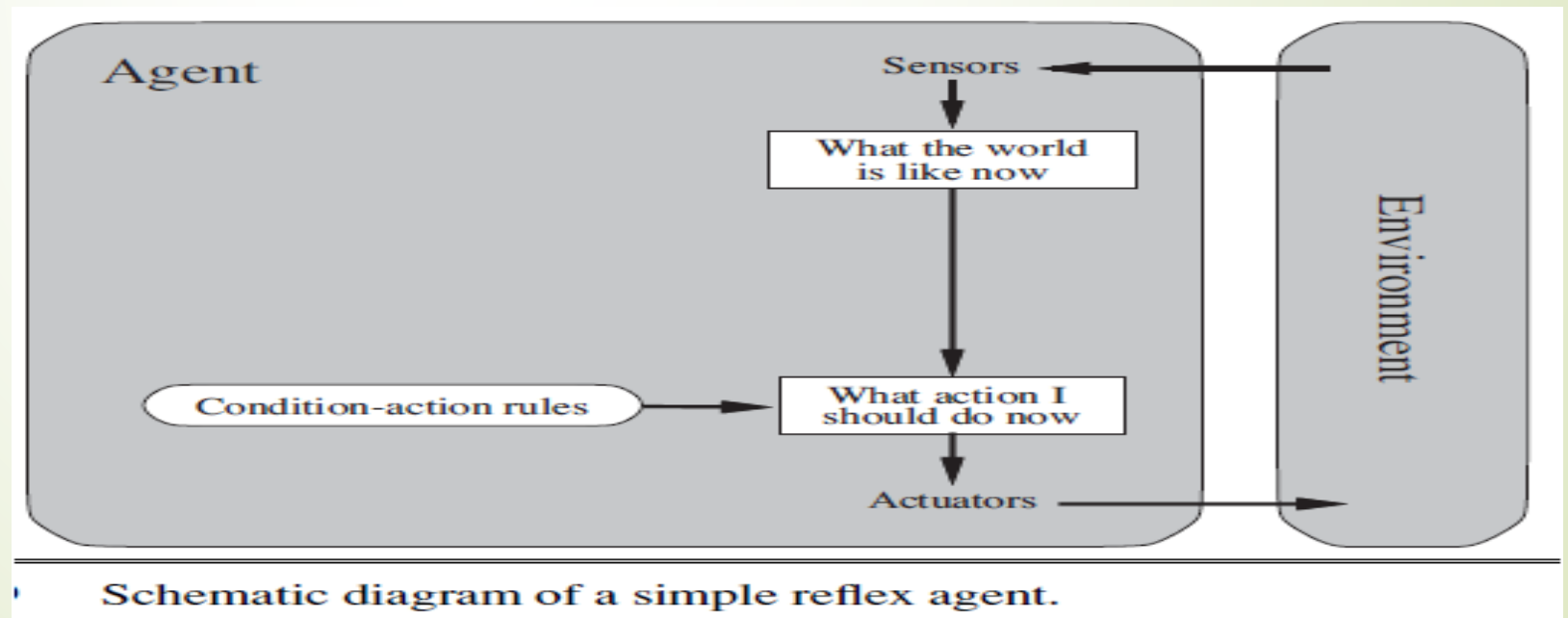
- The simplest kind of agent is the **simple reflex agent**.
- These agents select actions on the basis of the *current* percept, ignoring the rest of the percept history.
- For example, the vacuum agent whose agent function is tabulated in Figure is a simple reflex agent, because its decision is based only on the current location and on whether that location contains dirt.

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
⋮	⋮
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
⋮	⋮

- An agent program for this agent is shown in Figure

```
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 behaviors occur even in more complex environments.
- Imagine yourself as the driver of the automated taxi. If the car in front brakes and its brake lights come on, then you should notice this and initiate braking.
- some processing is done on the visual input to establish the condition we call “The car in front is braking.” Then, this triggers some established connection in the agent program to the action “initiate braking.”
- We call such a connection a **condition–action rule**, written as **if *car-in-front-is-braking* then *initiate-braking***.
- Figure, showing how the condition–action rules allow the agent to make the connection from percept to action.



- We use **rectangles** to denote the current internal state of the agent's decision process, and **ovals** to represent the background information used in the process.
- The agent program, which is also very simple, is shown in Figure.

```
function SIMPLE-REFLEX-AGENT(percept) returns an action  
persistent: rules, a set of condition–action rules  
  
  state ← INTERPRET-INPUT(percept)  
  rule ← RULE-MATCH(state, rules)  
  action ← rule.ACTION  
  return action
```

- The INTERPRET-INPUT function generates an abstracted description of the current state from the percept, and the RULE-MATCH function returns the first rule in the set of rules that matches the given state description.
- The agent in Figure will work *only if the correct decision can be made on the basis of only the current percept—that is, only if the environment is fully observable.*
- Even a little bit of unobservability can cause serious trouble.
- For example, the braking rule given earlier assumes that the condition *car-in-front-is-braking* can be determined from the current percept—a single frame of video.
- This works if the car in front has a centrally mounted brake light.

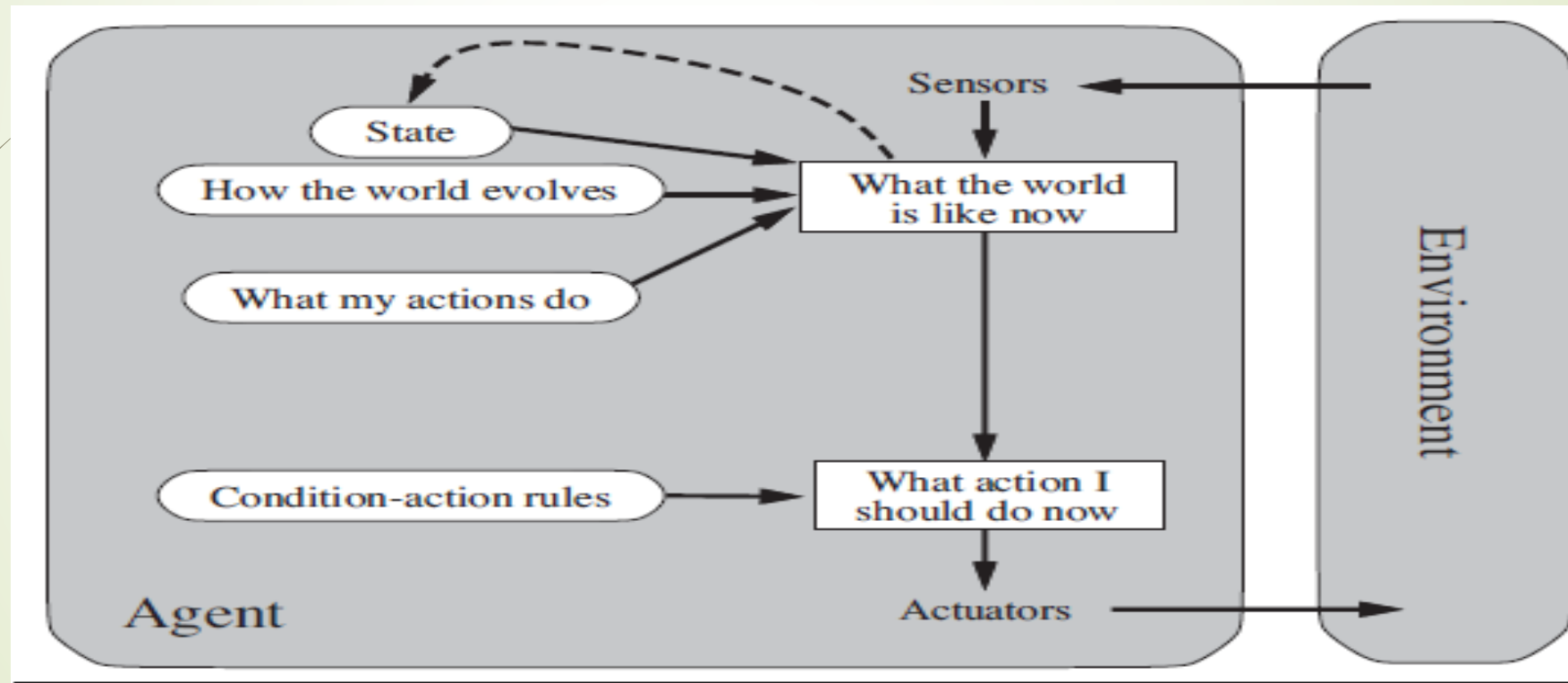
- We can see a similar problem arising in the vacuum world. Suppose that a simple reflex vacuum agent is deprived of its location sensor and has only a dirt sensor. Such an agent has just two possible percepts: [Dirty] and [Clean].
- It can Suck in response to [Dirty]; what should it do in response to [Clean]? Moving Left fails (forever) if it happens to start in square A, and moving Right fails (forever) if it happens to start in square B.
- Infinite loops are often unavoidable for simple reflex agents operating in partially observable environments.
- Escape from infinite loops is possible if the agent can **randomize** its actions.
- For example, if the vacuum agent perceives [Clean], it might flip a coin to choose between Left and Right . It is easy to show that the agent will reach the other square in an average of two steps.
- Then, if that square is dirty, the agent will clean it and the task will be complete.

Model-based reflex agents



The most effective way to handle partial observability is for the agent to *keep track of the part of the world it can't see now*.

- the agent should maintain some sort of **internal state** that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state.
- For the braking problem, the internal state is not too extensive just the previous frame from the camera, allowing the agent to detect when two red lights at the edge of the vehicle go on or off simultaneously.
- For other driving tasks such as changing lanes, the agent needs to keep track of where the other cars are if it can't see them all at once.
- Updating this internal state information as time goes by 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—for example, that an overtaking car generally will be closer behind than it was a moment ago.
- Second, we need some information about how the agent's own actions affect the world—for example, that when the agent turns the steering wheel clockwise, the car turns to the right, or that after driving for five minutes northbound on the freeway, one is usually about five miles north of where one was five minutes ago.

- ▶ “how the world works”—whether implemented in simple Boolean circuits or in complete scientific theories—is called a **model** of the world.
- ▶ An agent that uses such a model is called a **model-based agent**.
- ▶ Figure gives the structure of the model-based reflex agent with internal state, showing how the current percept is combined with the old internal state to generate the updated description of the current state, based on the agent’s model of how the world works

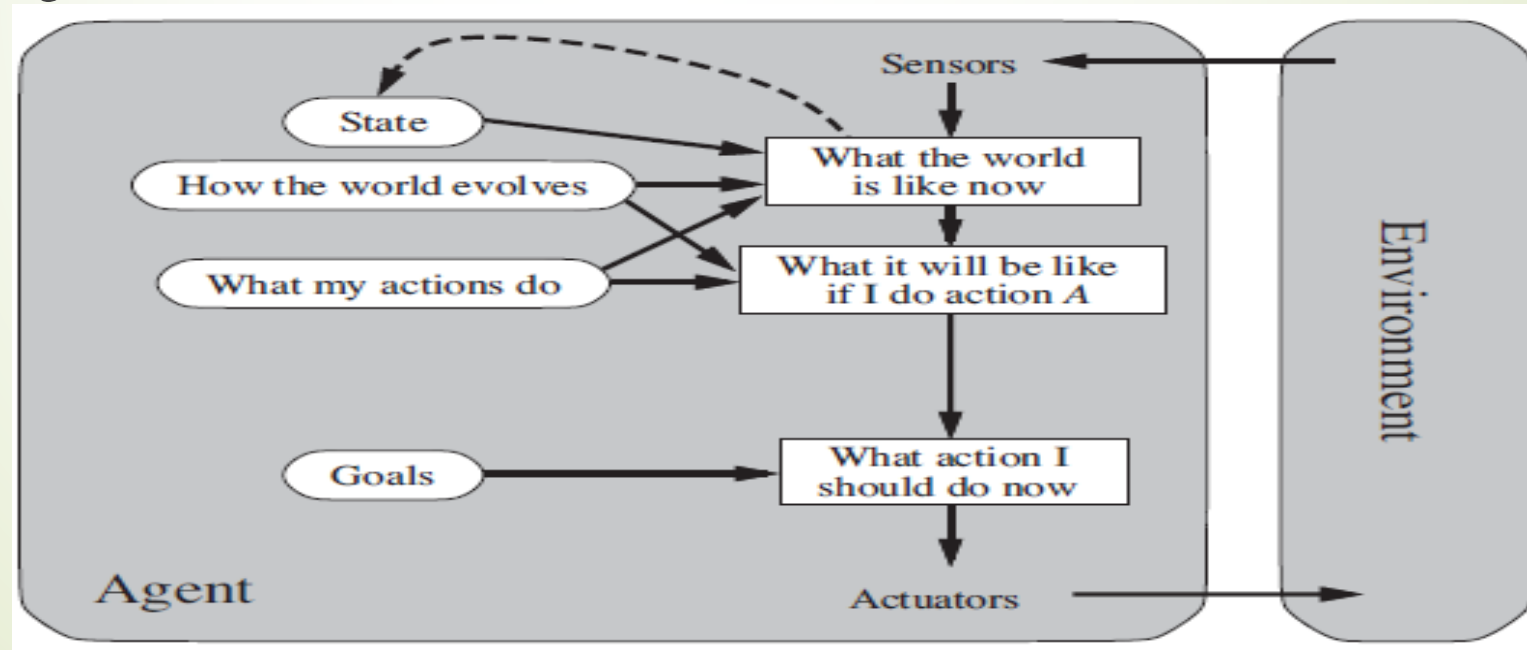


11 A model-based reflex agent.

- 
- 
- The interesting part is the function UPDATE-STATE, which is responsible for creating the new internal state description.
 - The details of how models and states are represented vary widely depending on the type of environment and the particular technology used in the agent design.
 - Regardless of the kind of representation used, it is seldom possible for the agent to determine the current state of a partially observable environment *exactly*.
 - the box labeled “what the world is like now” represents the agent’s “best guess” (or sometimes best guesses).
 - For example, an automated taxi may not be able to see around the large truck that has stopped in front of it and can only guess about what may be causing the hold-up.
 - Thus, uncertainty about the current state may be unavoidable, but the agent still has to make a decision.

Goal-based agents

- Knowing something about the current state of the environment is not always enough to decide what to do.
- For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to.
- In other words, as well as a current state description, the GOAL agent needs some sort of **goal** information that describes situations that are desirable. for example, being at the passenger's destination.
- The agent program can combine this with the model to choose actions that achieve the goal.



- Sometimes goal-based action selection is straightforward—for example, when goal satisfaction results immediately from a single action. Sometimes it will be more tricky.

For example, when the agent has to consider long sequences of twists and turns in order to find a way to achieve the goal.

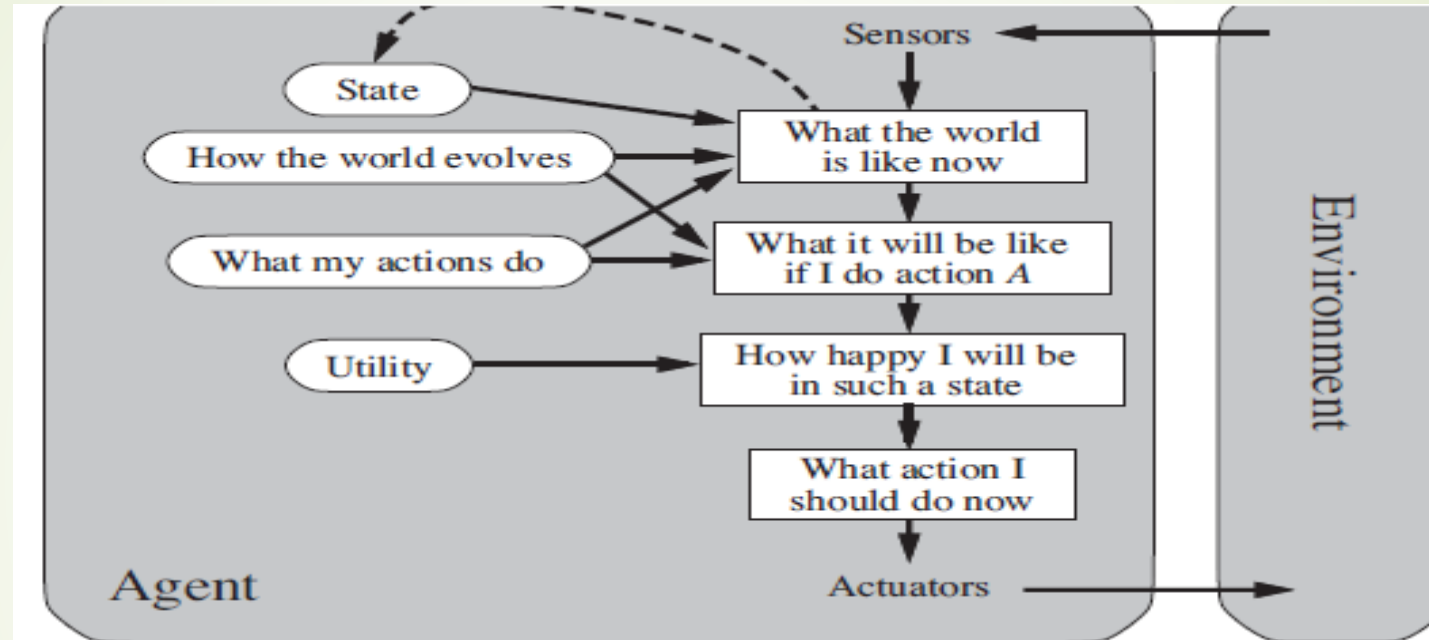
- **Search** and **planning** are the subfields of AI devoted to finding action sequences that achieve the agent's goals.
- it involves consideration of the future—both “What will happen if I do such-and-such?” and “Will that make me happy?”
- A goal-based agent, could reason that if the car in front has its brake lights on, it will slow down.

Utility-based agents

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

- Goals just provide a crude binary distinction between “happy” and “unhappy” states.
- A more general performance measure should allow a comparison of different world states according to exactly how happy they would make the agent.
- Because “happy” does not sound very scientific, economists and computer scientists use the term **utility** instead.
- An agent’s **utility function** is essentially an internalization of the performance measure.
- a utility-based agent has many advantages in terms of flexibility and learning. a utility-based agent can still make rational decisions.
- First, when there are conflicting goals, only some of which can be achieved (for example, speed and safety), the utility function specifies the appropriate tradeoff.
- Second, when 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.

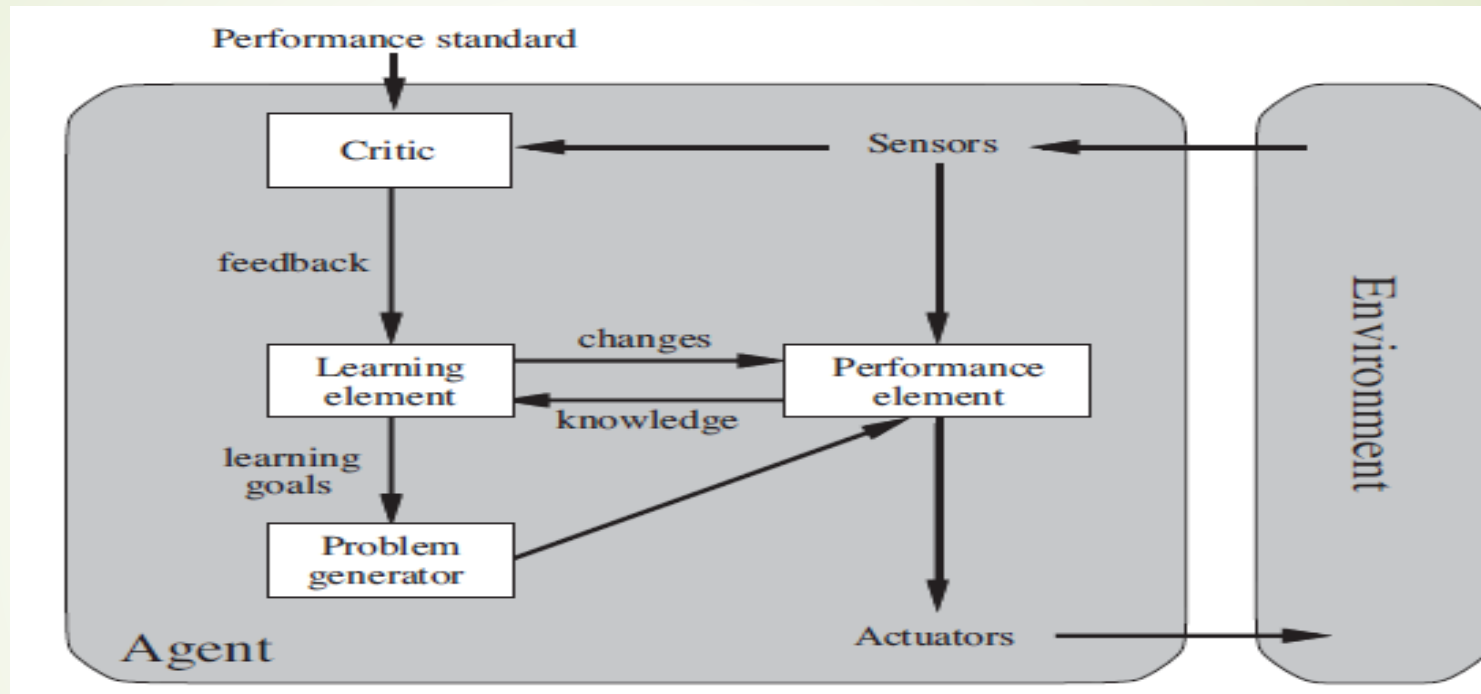
- The utility-based agent structure appears in Figure. Utility-based agent programs appear in Part IV, where we design decision-making agents that must handle the uncertain inherent in stochastic or partially observable environments.



- A utility-based agent has to model and keep track of its environment, tasks that have involved a great deal of research on perception, representation, reasoning, and learning.
- Choosing the utility-maximizing course of action is also a difficult task, requiring ingenious algorithms.

Learning agents

A learning agent can be divided into four conceptual components, as shown in Figure.



- The most important distinction is between the **learning element**, which is responsible for making improvements, and the **performance element**, which is responsible for selecting external actions.
- The **performance element** is what we have previously considered to be the entire agent: it takes in percepts and decides on actions. The **learning element** uses feedback from the **critic** on how the agent is doing and determines how the performance element should be modified to do better in the future.

- When trying to design an agent that learns a certain capability, the first question is not “How am I going to get it to learn this?” but “What kind of performance element will my agent need to do this once it has learned how?”
- The last component of the learning agent is the **problem generator**. It is responsible for suggesting actions that will lead to new and informative experiences.

➤ How the components of agent programs work

- **atomic, factored, and structured.** This component describes the changes that might occur in the environment as the result of taking an action

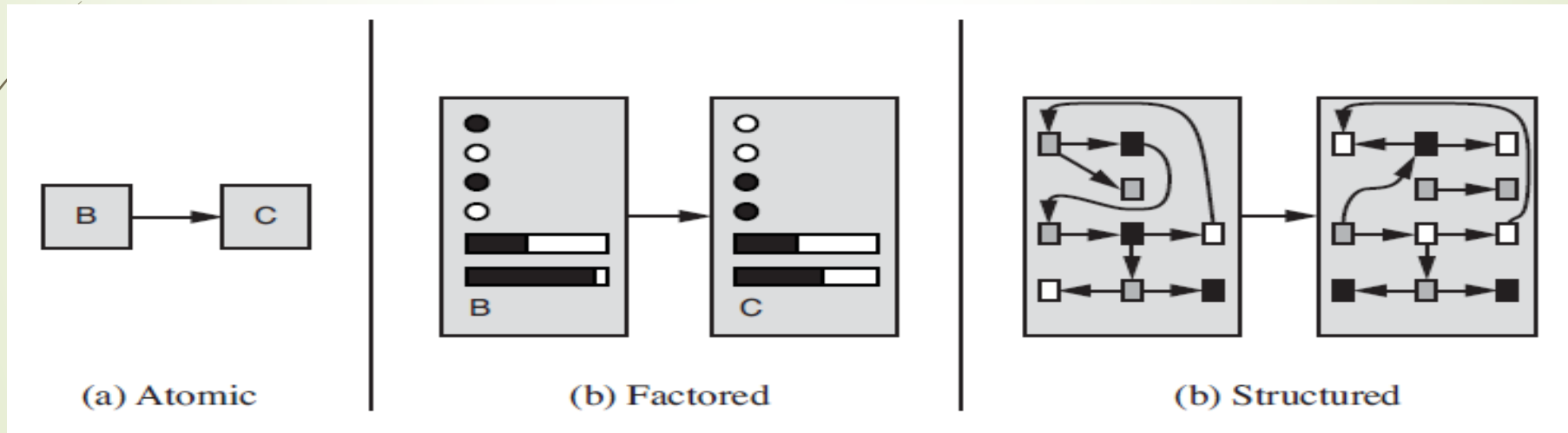



Figure 2.16 Three ways to represent states and the transitions between them. (a) Atomic representation: a state (such as B or C) is a black box with no internal structure; (b) Factored representation: a state consists of a vector of attribute values; values can be Boolean, real-valued, or one of a fixed set of symbols. (c) Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects.

- In an **atomic representation** each state of the world is indivisible—it has no internal structure.
- Consider the problem of finding a driving route from one end of a country to the other via some sequence of cities.
- For the purposes of solving this problem, it may suffice to reduce the state of world to just the name of the city we are in—a single atom of knowledge; a “black box” whose only discernible property is that of being identical to or different from another black box.
- The algorithms underlying **search** and **game-playing**, **Hidden Markov models** and **Markov decision processes** all work with atomic representations—or, at least, they treat representations *as if* they were atomic.
- A **factored representation** splits up each state into a fixed set of **variables** or **attributes**, each of which can have a **value**. While two different atomic states have nothing in common—they are just different black boxes—two different factored states can share some attributes and not others; this makes it much easier to work out how to turn one state into another.
- With factored representations, we can also represent *uncertainty*—for example, ignorance about the amount of gas in the tank can be represented by leaving that attribute blank.
- Many important areas of AI are based on factored representations, including **constraint satisfaction** algorithms, **propositional logic**, **planning**, **Bayesian networks**, and the **machine learning** algorithms.

- 
- a **structured representation**, in which objects such as cows and trucks and their various and varying relationships can be described explicitly.
 - Structured representations underlie **relational databases** and **first-order logic**, **first-order probability models**, **knowledge-based learning** and much of **natural language understanding**.



➤ **Do Exercise.**