Intelligent Agents

Course Code: **CSC4226** Course Title: **Artificial Intelligence and Expert System**

Dept. of Computer Science
Faculty of Science and Technology

Lecture No:	**Two (2)**	**Week No:**	**Two (2)**	**Semester:**	
Lecturer:	*Dr. Abdus Salam*		*Mail: abdus.salam@aiub.edu*		

Lecture Outline

1. Agents and Environments
2. Good Behavior: The Concept of Rationality
3. The Nature of Environments
4. The Structure of Agents

AGENT

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

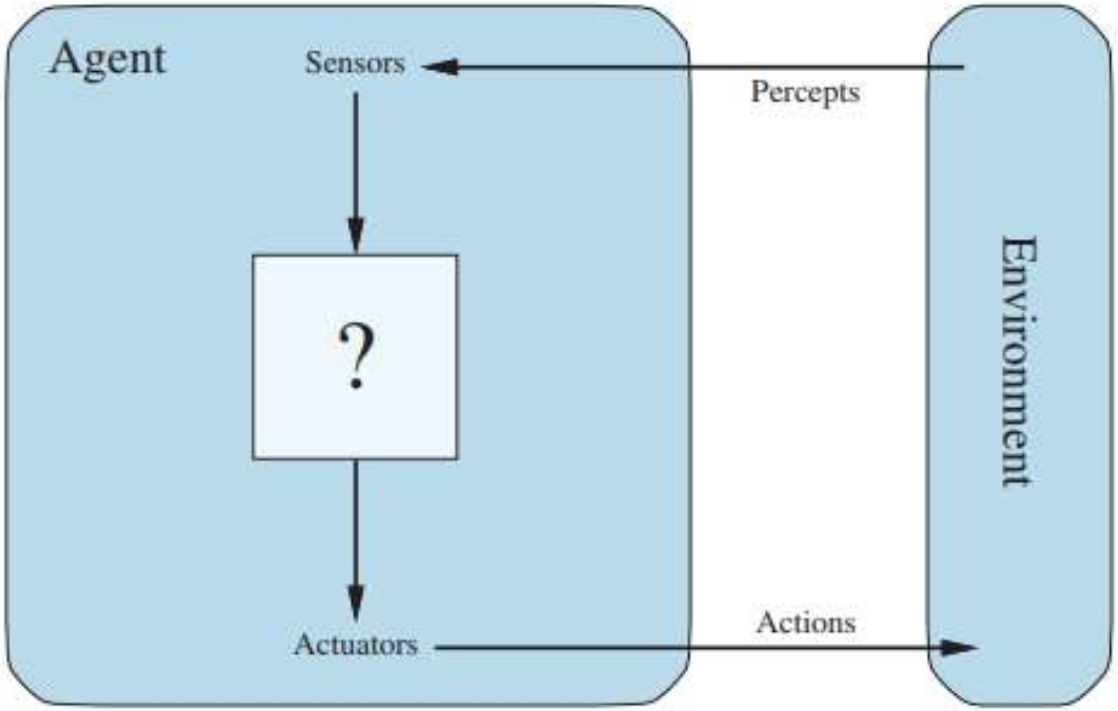


Figure 2.1 Agents interact with environments through sensors and actuators.

INTELLIGENT AGENT

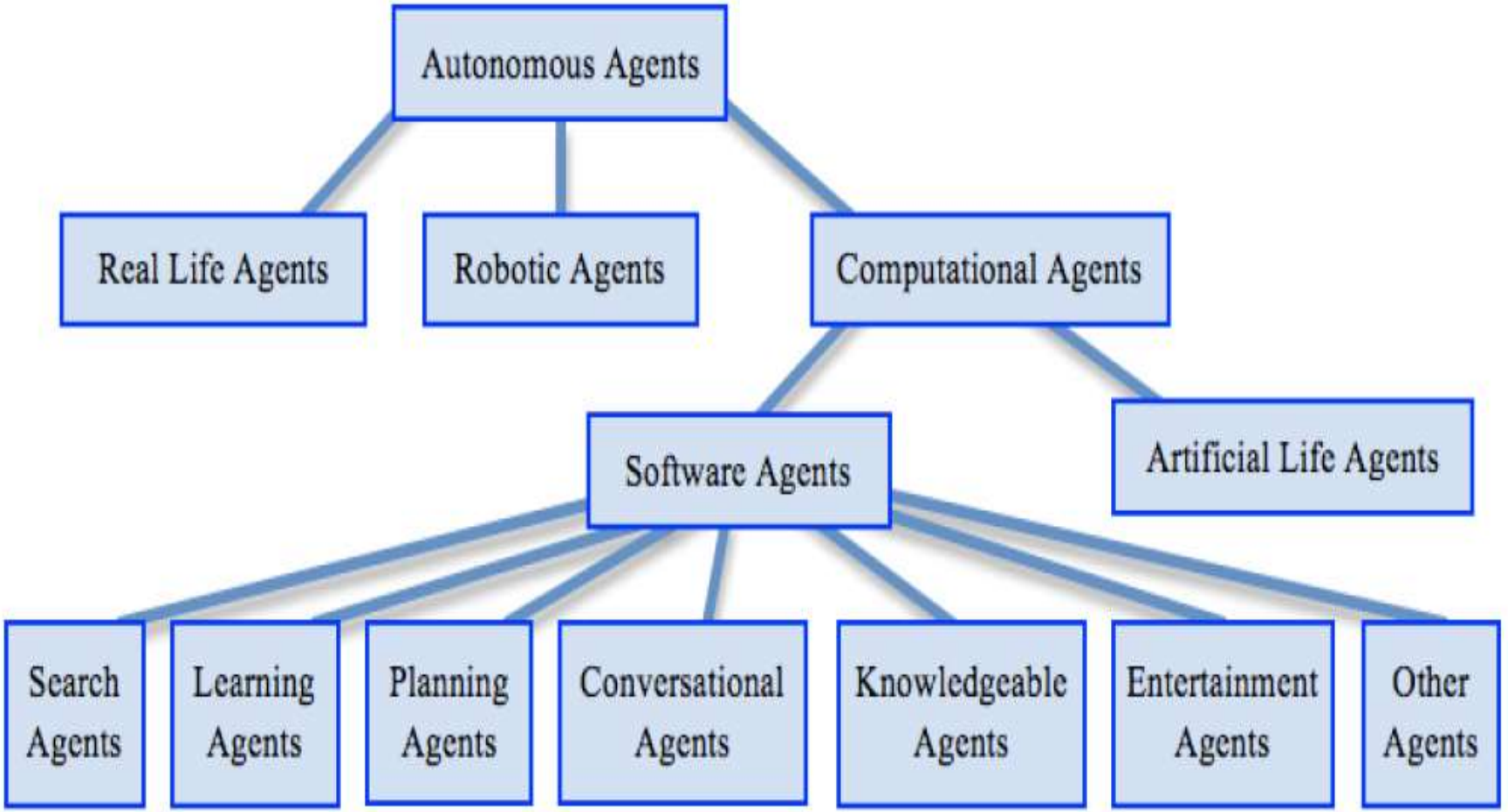
Agent: An entity in a program or environment capable of generating action.

An agent uses perception of the environment to make decisions about actions to take.

The perception capability is usually called a sensor.

The actions can depend on the most recent perception or on the entire history (percept sequence).

TAXONOMY OF AUTONOMOUS AGENT



AGENT V/S PROGRAM

Size - an agent is usually smaller than a program.

Purpose - an agent has a specific purpose while programs are multi-functional.

Persistence - an agent's life span is not entirely dependent on a user launching and quitting it.

Autonomy - an agent doesn't need the user's input to function.

DIFFERENT AGENTS

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.

AGENTS

The term **percept** refers 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.

In general, an agent's choice of **action** at any given instant can depend on the entire **percept sequence** observed to date, but not on anything it hasn't perceived.

AGENT FUNCTION

The agent's choice of action for every possible percept sequence can be defined.

An agent's behavior is described by the **agent function** that maps any given percept sequence to an action.

We can imagine tabulating the agent function that describes any given agent; for most agents, this would be a very large table—infinite, in fact, unless we place a bound on the length of percept sequences we want to consider.

AGENT PROGRAM

The table can be constructed by trying out all possible percept sequences and recording which actions the agent does in response. Thus, the table is, of course, an external characterization of the agent. Internally, 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.

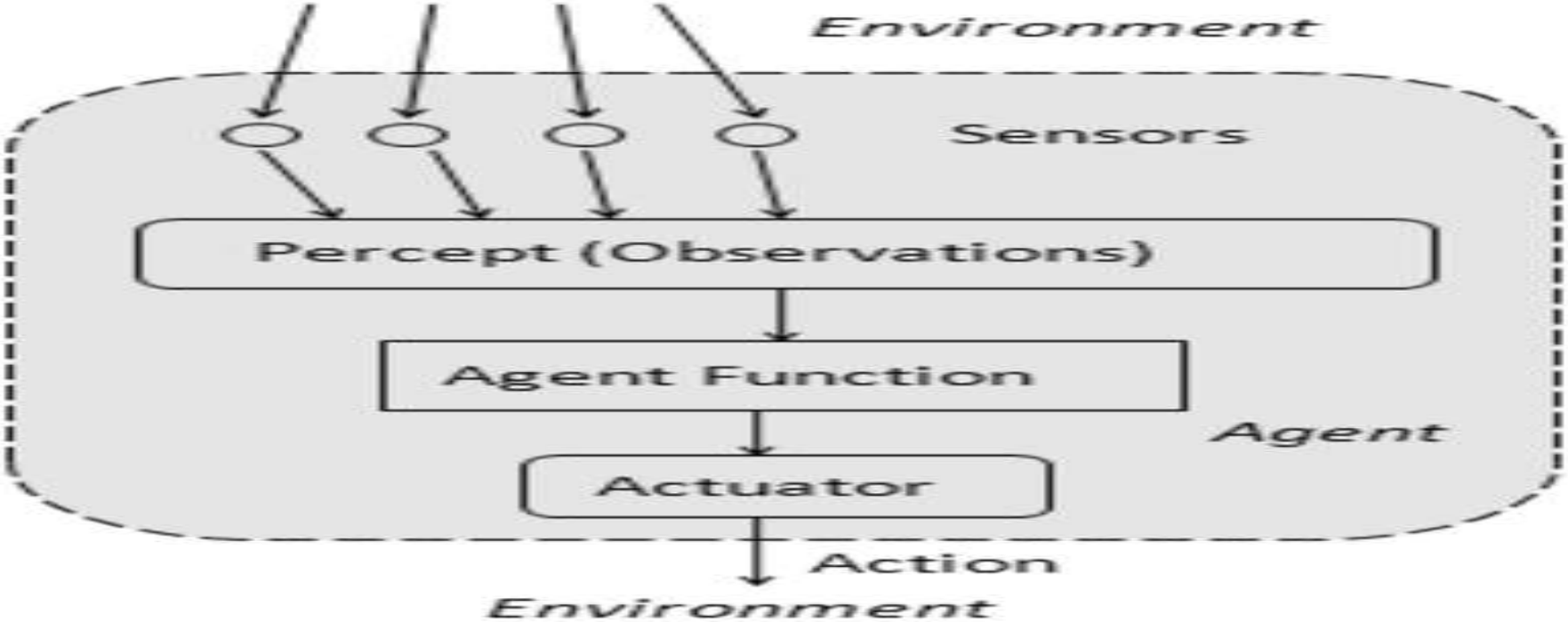
AGENT FUNCTION

The agent function is a mathematical function that maps a sequence of perceptions into action.

The **function** is implemented as the **agent program**.

The part of the agent taking an action is called an actuator.

environment -> sensors -> agent function -> actuators -> environment



VACUUM CLEANING AGENT

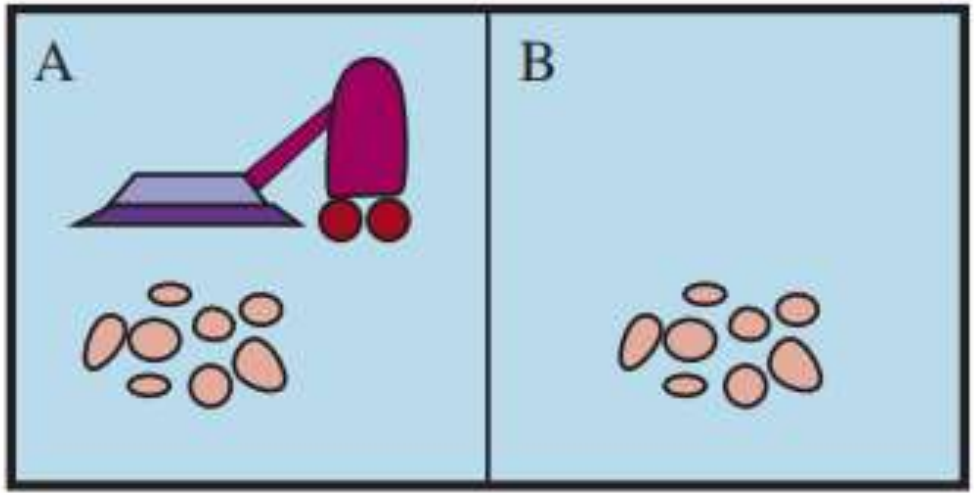


Figure 2.2 A vacuum-cleaner world with just two locations. Each location can be clean or dirty, and the agent can move left or right and can clean the square that it occupies. Different versions of the vacuum world allow for different rules about what the agent can perceive, whether its actions always succeed, and so on.

VACUUM CLEANING AGENT

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*
⋮	⋮

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure **??**. The agent cleans the current square if it is dirty, otherwise it moves to the other square. Note that the table is of unbounded size unless there is a restriction on the length of possible percept sequences.

VACUUM CLEANING AGENT

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.

DESIRABLE PROPERTIES OF AGENT

Property	Description
Autonomy	The agent exercises control over its own actions; it runs asynchronously.
Reactivity	The agent responds in a timely fashion to changes in the environment and decides for itself when to act.
Proactivity	The agent responds in the best possible way to possible future actions that are anticipated to happen.
Social ability (Ability to communicate)	The agent has the ability to communicate in a complex manner with other agents, including people, in order to obtain information or elicit help in achieving its goals.
Ability to set goals	The agent has a purpose.
Temporal continuity	The agent is a continually running process.

DESIRABLE PROPERTIES OF AGENT

Continued..

Property	Description
Mobility	The agent is able to transport itself around its environment.
Adaptivity	The agent has the ability to learn. It is able to change its behaviour based on the basis of its previous experience.
Benevolence	The agent performs its actions for the benefit of others.
Rationality	The agent makes rational, informed decisions.
Collaborative ability	The agent collaborates with other agents or humans to perform its tasks.
Flexibility	The agent is able to dynamically respond to the external state of the environment by choosing its own actions.
Personality	The agent has a well-defined, believable personality and emotional state.
Cognitive ability	The agent is able to explicitly reason about its own intentions or the state and plans of other agents.
Versatility	The agent is able to have multiple goals at the same time.
Veracity	The agent will not knowingly communicate false information.
Persistency	The agent will continue steadfastly in pursuit of any plan.

DESIRABLE PROPERTIES OF AGENT

Continued..

Property	Description
Coordination	The agent has the ability to manage resources when they need to be distributed or synchronised.
Cooperation	The agent makes use of interaction protocols beyond simple dialogues, for example negotiations on finding a common position, solving conflicts or distributing tasks
Ability to plan	The agent has the ability to pro-actively plan and coordinating its reactive behavior in the presence of the dynamical environment formed by the other acting agents.

GOOD BEHAVIOR:
THE CONCEPT OF RATIONALITY

Rational Agent

- one does the right thing
- Every entry in the table for the agent function is filled out correctly

What does it mean to do the right thing ?

by considering the *consequences* of the agent's behavior

When an agent is plunked down in an **environment**, it generates a sequence of **actions** according to the **percepts** it receives.

This sequence of **actions** causes the **environment** to go through a sequence of **states**. If the sequence is desirable, then the agent has performed well.

This notion of desirability is captured by a **performance measure** that evaluates any given sequence of environment states.

RATIONAL AGENT

A rational agent is one that can take the right decision in every situation.

Performance measure: a set of criteria/test bed for the success of the agent's behavior.

The performance measures should be based on the desired effect of the agent on the environment.

RATIONALITY

The agent's rational behavior depends on:

- the **performance measure** that **defines success**
- the agent's **knowledge** of the **environment**
- the **action** that it is capable of **performing**
- the **current sequence** of **perceptions**.

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.

SPECIFYING THE TASK ENVIRONMENT:

PEAS DESCRIPTION

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits, minimize impact on other road users	Roads, other traffic, police, pedestrians, customers, weather	Steering, accelerator, brake, signal, horn, display, speech	Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen

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

PEAS: Examples

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.

PROPERTIES OF TASK ENVIRONMENT

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; relevance. 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**.

PROPERTIES OF TASK ENVIRONMENT

We say an environment is **uncertain** if it is not fully observable or not deterministic.

In our use of the word "stochastic" generally implies that uncertainty about outcomes is quantified in terms of probabilities; a **nondeterministic** environment is one in which actions are characterized by their *possible* outcomes, but no probabilities are attached to them.

PROPERTIES OF TASK ENVIRONMENT

Single agent vs. **multi-agent**: The distinction between single-agent and multi-agent environments may seem simple enough. 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.

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.

PROPERTIES OF TASK ENVIRONMENT

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.

PROPERTIES OF TASK ENVIRONMENT

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, on the other hand, are continuously asking the agent what it wants to do; if it hasn't decided yet, that counts as deciding to do nothing.

PROPERTIES OF TASK ENVIRONMENT

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.).
Input from digital cameras is discrete, strictly speaking, but is typically treated as representing continuously varying intensities and locations.

TASK ENVIRONMENT: EXAMPLES

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Figure 2.6 Examples of task environments and their characteristics.

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

Obviously, the program we choose has to be one that is appropriate for the architecture. If the program is going to recommend actions like **Walk**, the architecture had better have **legs**. The architecture might be just an ordinary PC, or it might be a robotic car with several onboard computers, cameras, and other sensors.

AGENT EXAMPLE: TABLE DRIVEN AGENT

Table-driven agents: It consists in a lookup table of actions to be taken for every possible state of the environment.

If the environment has **n** variables, each with **t** possible states, then the table size is $\mathbf{t^n}$.

Only works for a small number of possible states for the environment.

TABLE-DRIVEN-AGENT

function TABLE-DRIVEN-AGENT(*percept*) **returns** an action
  persistent: *percepts*, a sequence, initially empty
    table, a table of actions, indexed by percept sequences, initially fully specified

  append *percept* to the end of *percepts*
  action ← LOOKUP(*percepts*, *table*)
  return *action*

Figure 2.7 The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.

VACUUM CLEANING AGENT: Table Driven

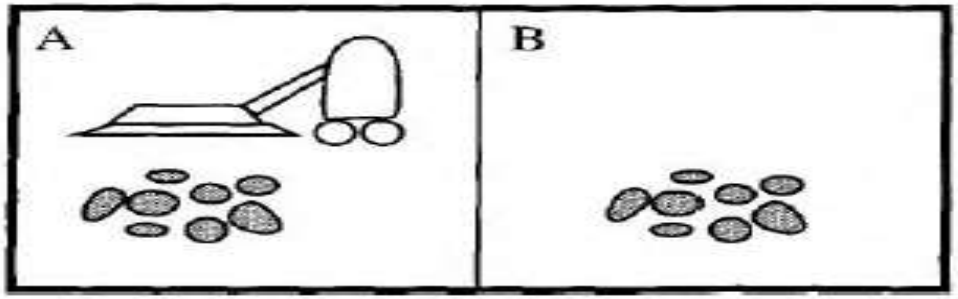


Figure 2.2 A vacuum-cleaner world with just two locations.

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

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

LIMITATION OF TABLE-DRIVEN AGENT

No physical agent in this universe will have the space to store the table,

No agent could ever learn all the right table entries from its experience, and

Even if the environment is simple enough to yield a feasible table size, the designer still has no guidance about how to fill in the table entries

BASIC KINDS OF AGENT PROGRAMS

1. Simple reflex agents
2. Model-based reflex agents
3. Goal-based agents
4. Utility-based agents.

SIMPLE REFLEX AGENTS

select actions on the basis of the *current percept, ignoring the rest of the percept history.*

a ***condition-action rule***, written as

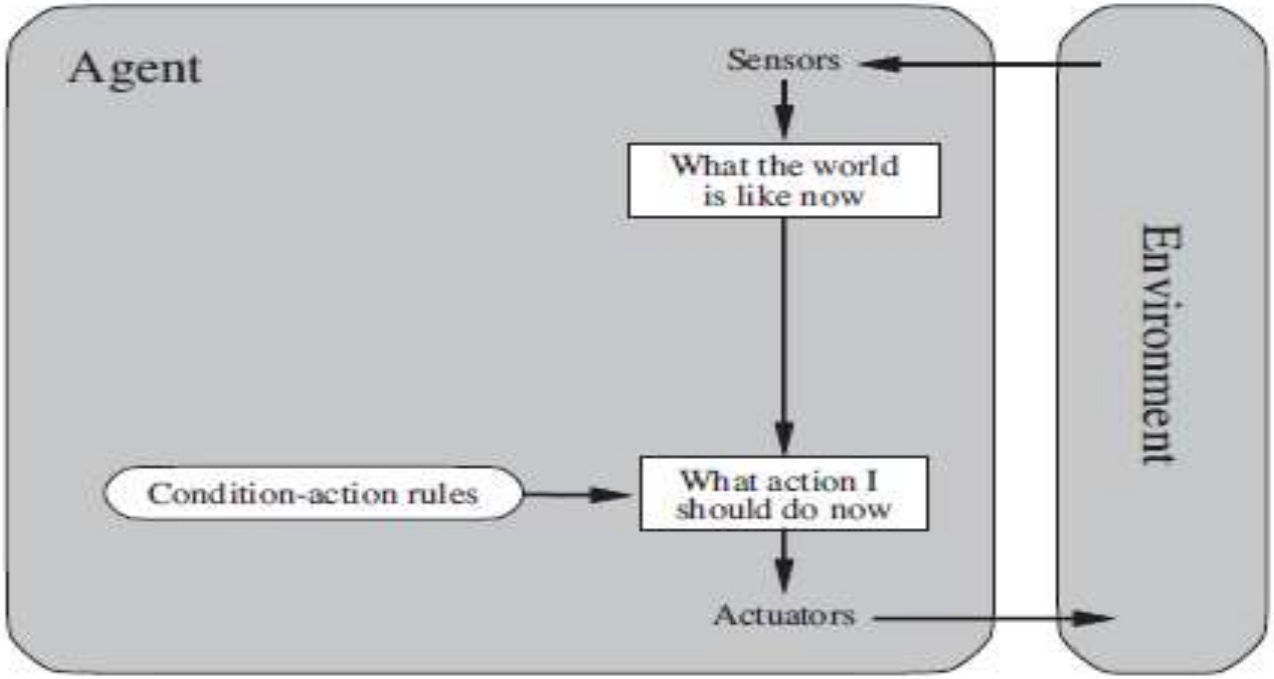
if *car-in-front-is-braking then initiate-braking.*

general and flexible approach is first to build a general-purpose interpreter for condition-action rules and then to create rule sets for specific task environments.

```
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.

Works only if the Environment is Fully Observable



Problem??
Infinite Loop in Deterministic SRA

Solution??
Randomized SRA

Figure 2.9 Schematic diagram of a simple reflex agent.

```
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
```

What will happen? when,
Vacuum Cleaner with poor perception!
Without location sensor !

Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

SIMPLE REFLEX AGENTS

Simple reflex agents have the admirable property of being simple, but they turn out to be of limited intelligence.

The agent in Figure 2.10 will work only if the correct decision can be made based on only the current percept—that is, only if the environment is fully observable.

Even a little bit of unobservability can cause serious trouble.

MODEL-BASED REFLEX AGENTS

*keep track of the part of the world it can't see now. [*handle partial observability]

That is, 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.

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.

MODEL-BASED REFLEX AGENTS

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.

This knowledge about "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**.

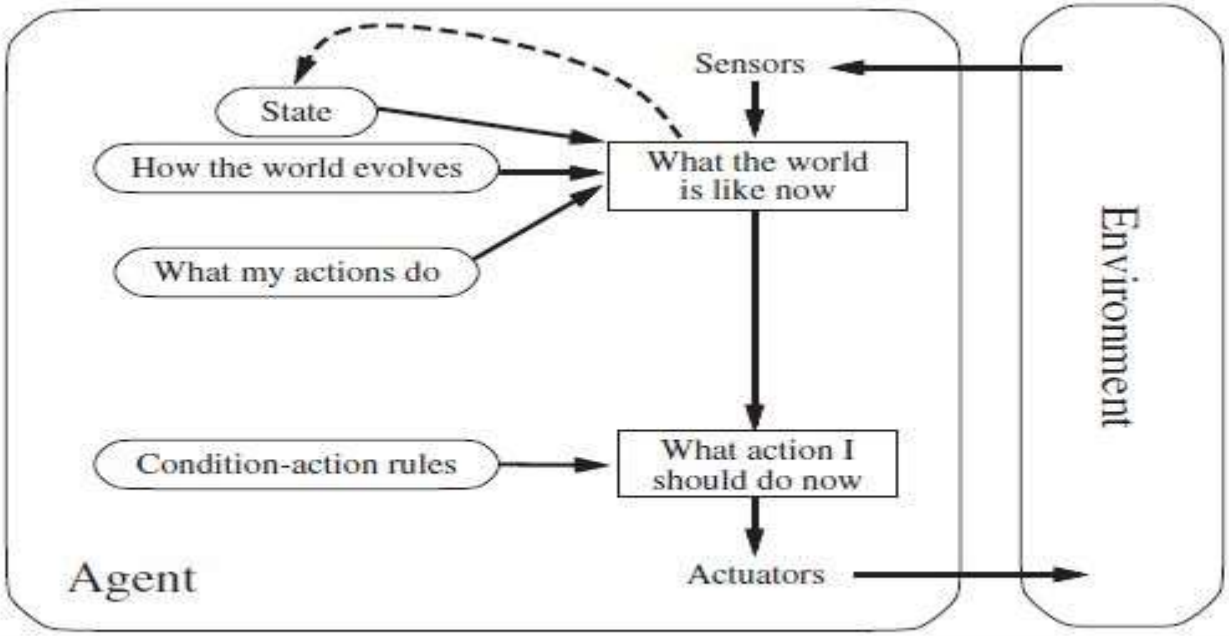


Figure 2.11 A model-based reflex agent.

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 ← UPDATE-STATE(*state*, *action*, *percept*, *model*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

Figure 2.12 A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

GOAL-BASED AGENT

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 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 (the same information as was used in the model-based reflex agent) to choose actions that achieve the goal.

GOAL-BASED AGENT

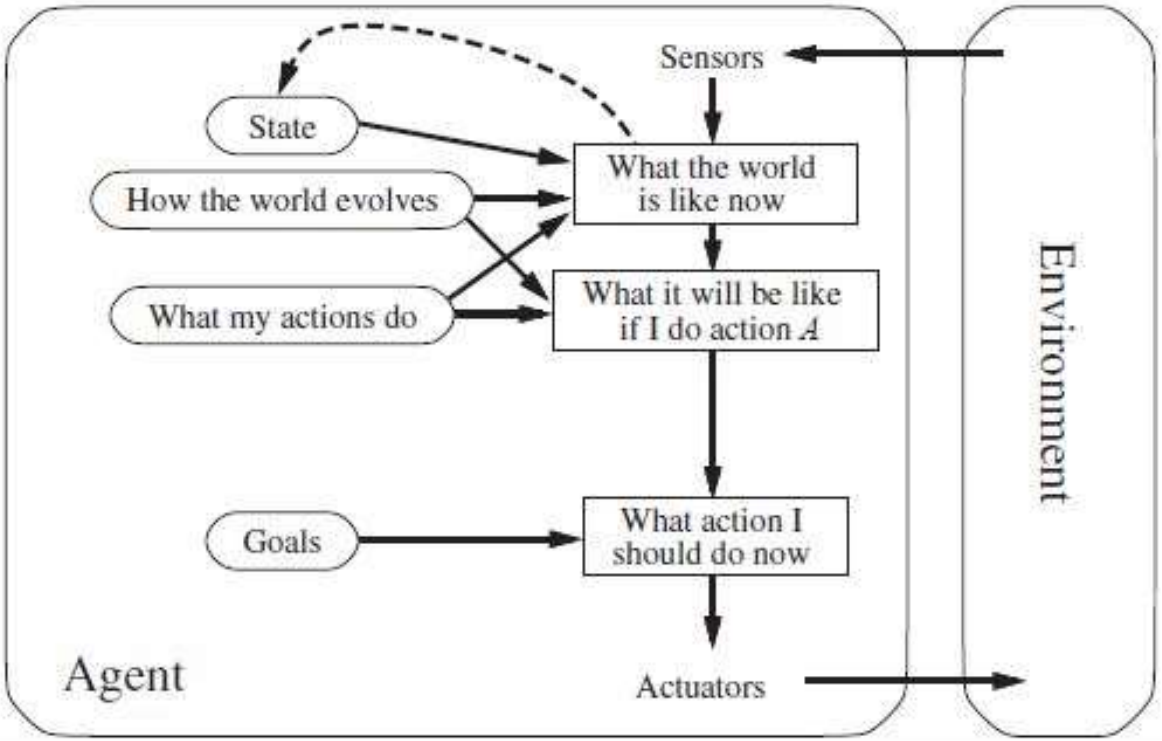


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

GOAL-BASED AGENT

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 (Chapters 3 to 5) and **planning** (Chapters 10 and 11) are the subfields of AI devoted to finding action sequences that achieve the agent's goals.

The goal-based agent's behavior can easily be changed to go to a different destination, simply by specifying that destination as the goal. **The reflex agent's** rules for when to turn and when to go straight will work only for a single destination; they must all be replaced to go somewhere new.

UTILITY-BASED AGENTS

Goals alone are not enough to generate high-quality behavior in most environments. For example, many action sequences will get the taxi to its destination (thereby achieving the goal) but **some are quicker, safer, more reliable, or cheaper than others**.

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.

UTILITY-BASED AGENTS

An agent's **utility function** is essentially an internalization of the performance measure. If the internal utility function and the external performance measure are in agreement, then an agent that chooses actions to maximize its utility will be rational according to the external performance measure.

In two kinds of cases, goals are inadequate, but a utility-based agent can still make rational decisions.

1. when there are conflicting goals, only some of which can *be achieved (for example, speed and safety), the utility function specifies the appropriate tradeoff.*
2. 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.

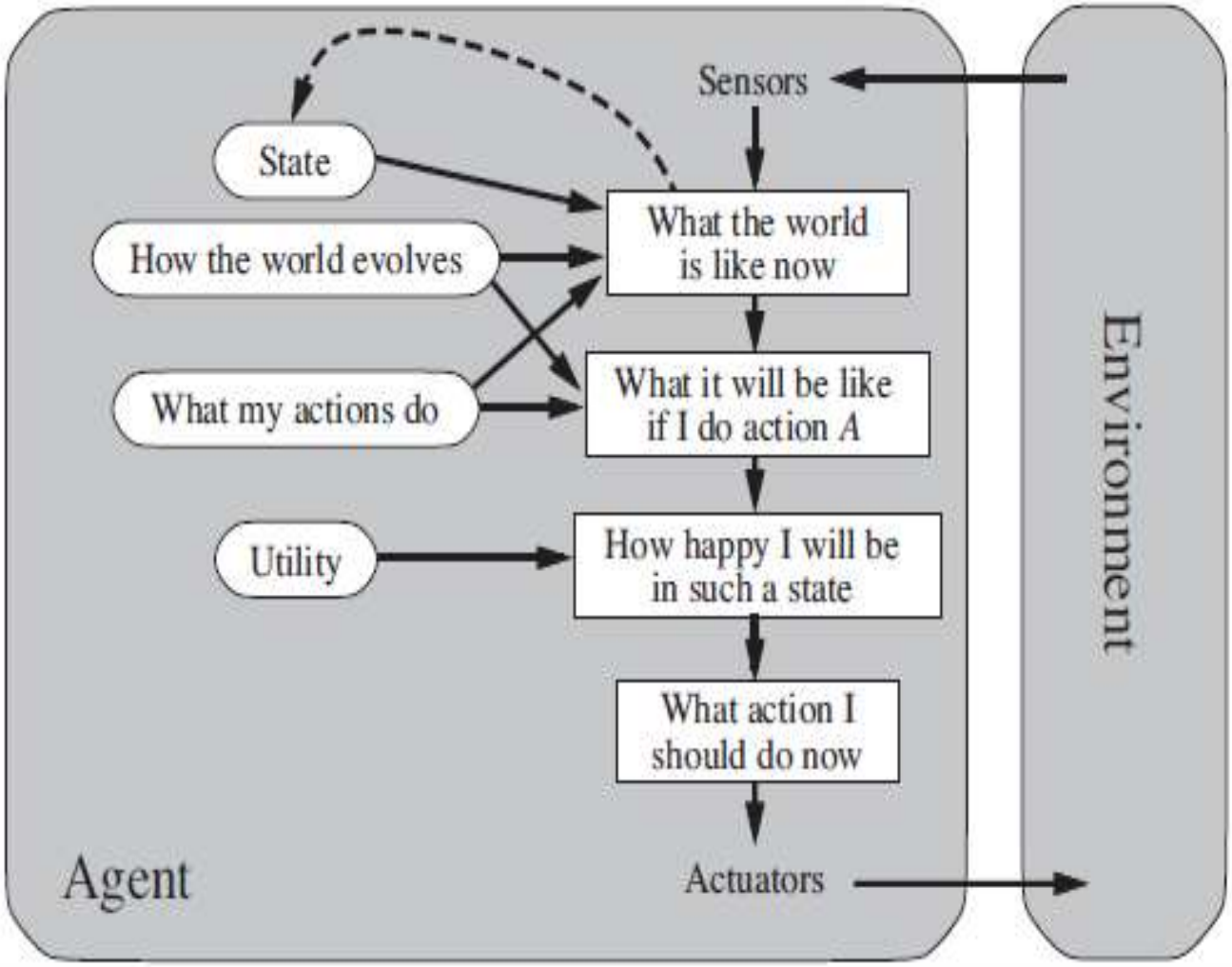


Figure 2.14 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome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

LEARNING AGENT

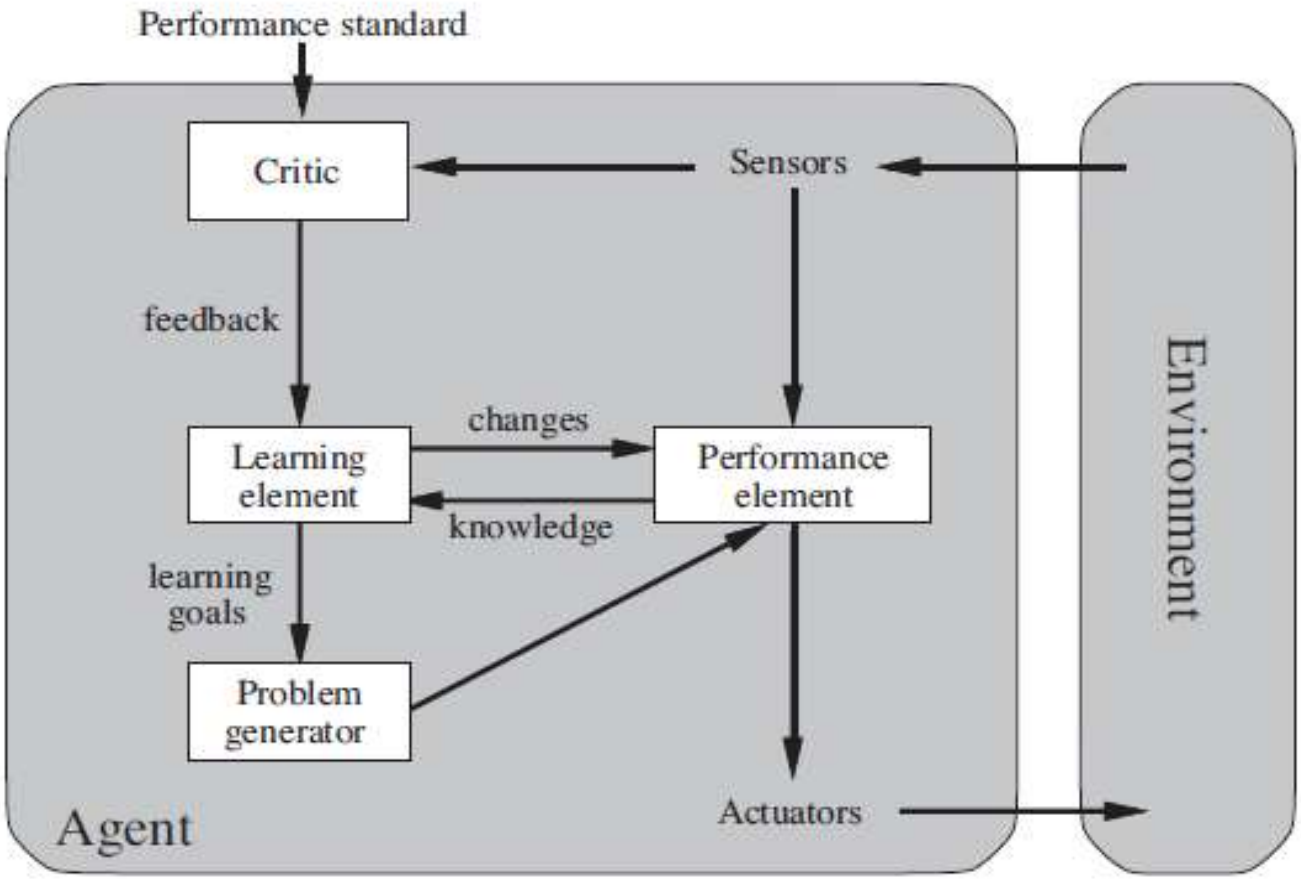


Figure 2.15 A general learning agent.

HOW COMPONENT OF AGENT PROGRAMS WORK

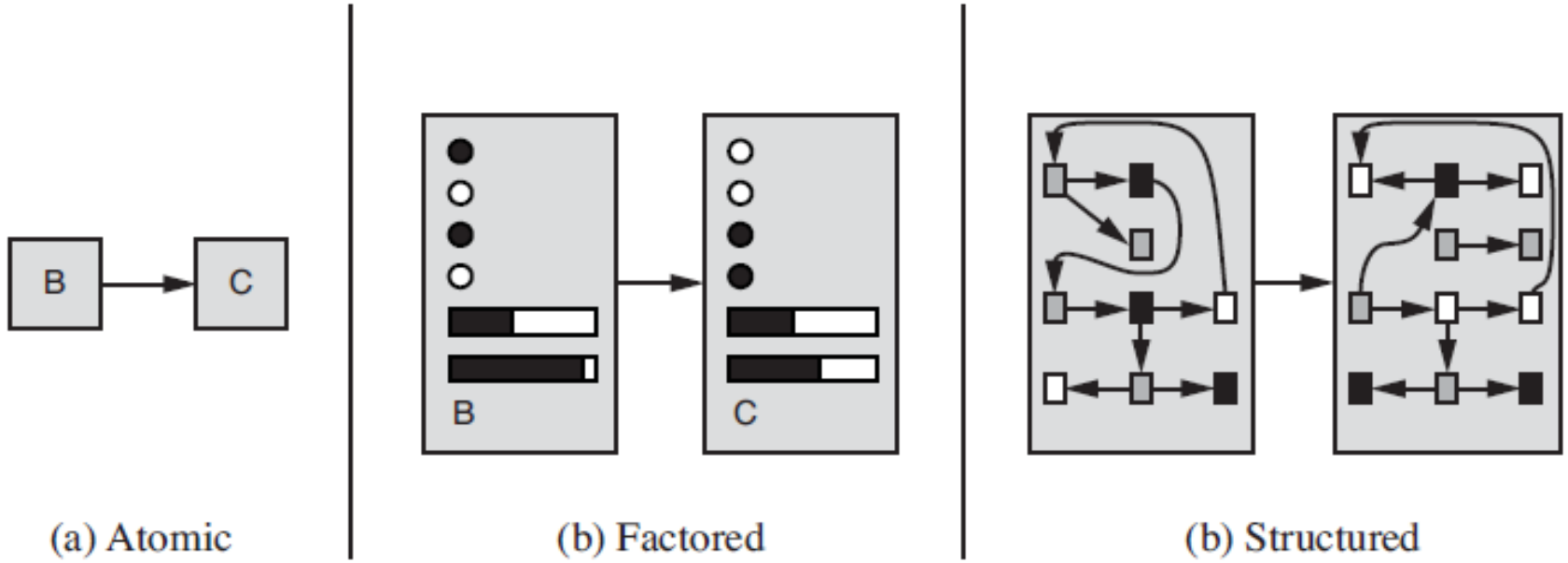


Figure 2.16 Three ways to represent states and the transitions between them. (a) Atomic representation: a state (such as B or C) is a black box with no internal structure; (b) Factored representation: a state consists of a vector of attribute values; values can be Boolean, real-valued, or one of a fixed 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.

References

1. Chapter 2: Intelligent Agents , Pages 34-58
“Artificial Intelligence: A Modern Approach,” by Stuart J. Russell and Peter Norvig,

Books

1. “Artificial Intelligence: A Modern Approach,” by Stuart J. Russell and Peter Norvig.
2. "Artificial Intelligence: Structures and Strategies for Complex Problem Solving", by George F. Luger, (2002)
3. "Artificial Intelligence: Theory and Practice", by Thomas Dean.
4. "AI: A New Synthesis", by Nils J. Nilsson.
5. “Programming for machine learning,” by J. Ross Quinlan,
6. “Neural Computing Theory and Practice,” by Philip D. Wasserman, .
7. “Neural Network Design,” by Martin T. Hagan, Howard B. Demuth, Mark H. Beale, .
8. “Practical Genetic Algorithms,” by Randy L. Haupt and Sue Ellen Haupt.
9. “Genetic Algorithms in Search, optimization and Machine learning,” by David E. Goldberg.
10. "Computational Intelligence: A Logical Approach", by David Poole, Alan Mackworth, and Randy Goebel.
11. “Introduction to Turbo Prolog”, by Carl Townsend.