# Chapter 2 - Intelligent Agents

Russell, S., & Norvig, P. (2022). *Artificial Intelligence - A Modern Approach* (4th global ed.). Pearson.

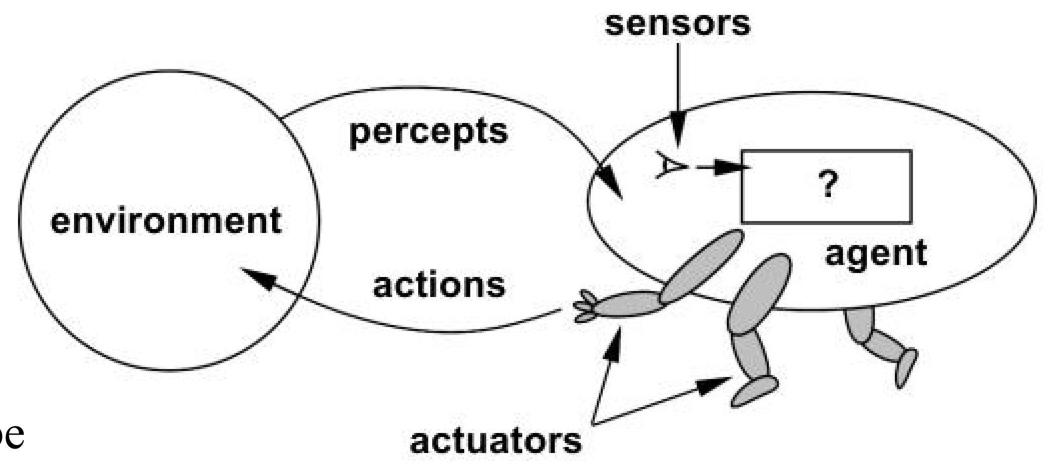
## Contents

## 1. Agent and Environments

- **Agents** include humans, robots, softbots, thermostats, etc.
- An **agent** can be anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**.
- The **agent function** maps any given percept sequence to an action:

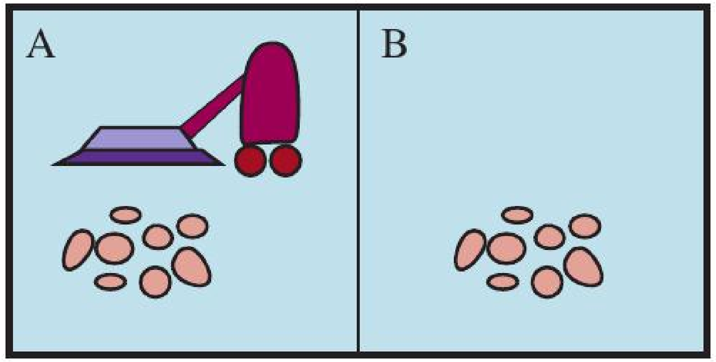
$$f: P^* \rightarrow A$$

- 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.



# 1. Agent and Environments

- A vacuum-cleaner world



  - Percepts: Location and Contents, e.g., [*A*, *Dirty*]
  - Actions: *Left, Right, Suck, NoOp*

| 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* |
| ⋮ | ⋮ |

- What is the right function?

- Can it be implemented in a small program?

## 2. Good Behavior: The Concept of Rationality

- A **rational agent** is one that does the right thing. Obviously, doing the right thing is better than doing the wrong thing, but what does it mean to do the right thing?
- **Performance Measures**
  - We evaluate an agent's behavior by its consequences.
  - 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.
  - *As a general rule, it is better to design performance measures according to what one actually wants to be achieved in the environment, rather than according to how one thinks the agent should behave.*

## 2. Good Behavior: The Concept of Rationality

- Fixed performance measure evaluates the environment sequence:
  - one point per square cleaned up in time $T$?
  - one point per clean square per time step, minus one per move?
  - penalize for $> k$ dirty squares?

## 2. 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.*

## 2. Good Behavior: The Concept of Rationality

- **Omniscience, Learning, and Autonomy**
  - An omniscient agent knows the *actual* outcome of its actions and can act accordingly; but omniscience is impossible in reality.
    - Rationality maximizes *expected* performance, while perfection maximizes *actual* performance.
  - A rational agent not only to gather information but also to **learn** as much as possible from what it perceives.
  - A rational agent should be **autonomous**—it should learn what it can to compensate for partial or incorrect prior knowledge.

## 3. The Nature of Environments
### Specifying the Task Environment

- To design a rational agent, we must specify the **task environment**.
- Task environment (PEAS description)
  - **P**erformance measure
  - **E**nvironment
  - **A**ctuators
  - **S**ensors
- The nature of the task environment directly affects the appropriate design for the agent program.

## 3. The Nature of Environments
## Specifying the Task Environment

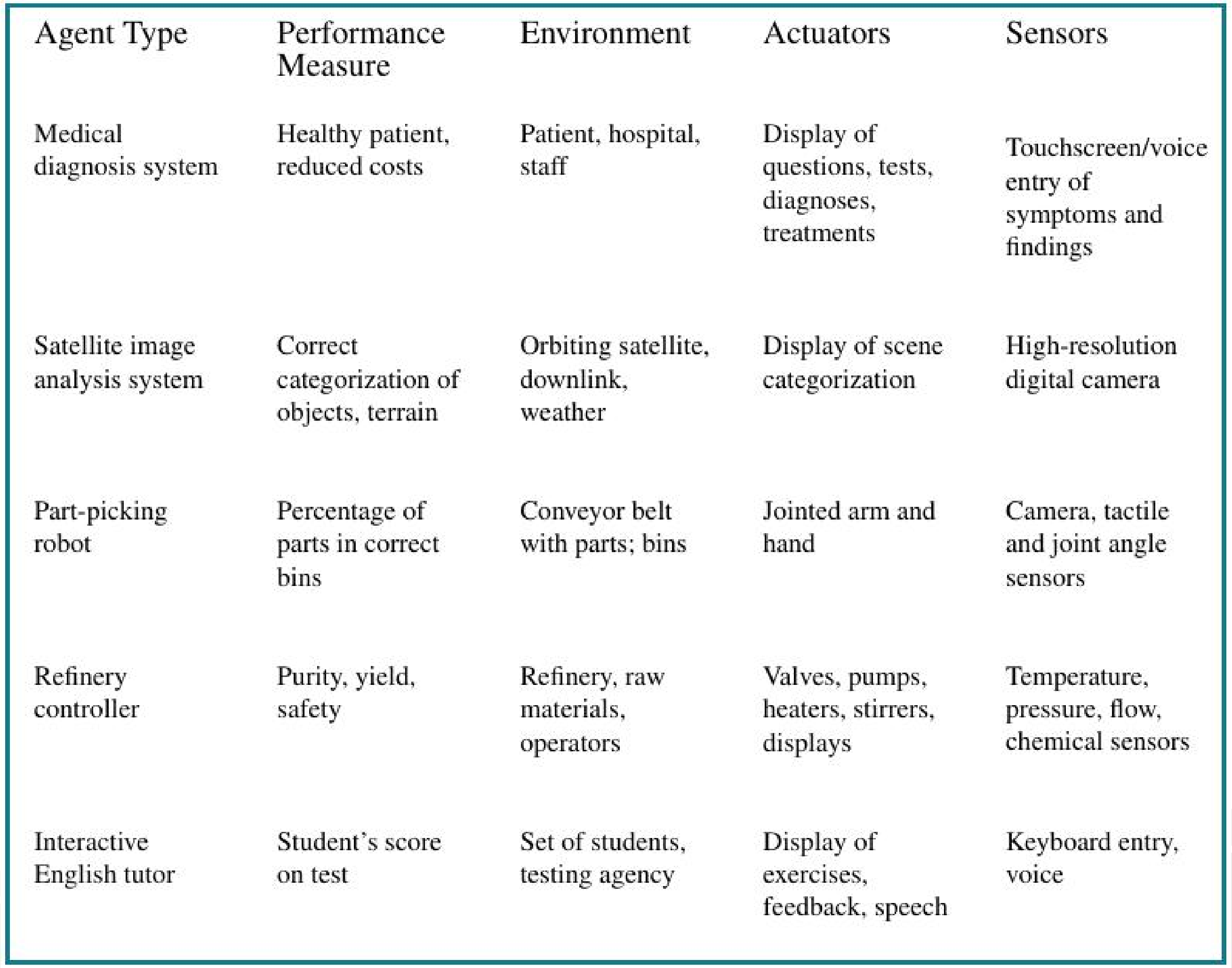
- PEAS description of the task environment for an automated taxi driver:
  - **P**erformance measure: Safe, fast, legal, comfortable, trip, maximize profit, minimize impat on other road users
  - **E**nvironment: Roads, other traffic, police, pedestrians, customer, weather
  - **A**ctuators: Steering, accelerator, brake, signal, horn, display, speed
  - **S**ensors: Cameras, radar, speepdometerm GPS, engine sensor, accelerometer, microphones, touchscreen

## 3. The Nature of Environments
## Specifying the Task Environment

- PEAS description of the task environment for an internet shopping agent:
  - **P**erformance measure: price, quality, appropriateness, efficiency
  - **E**nvironment: current WWW sites, vendors, shippers
  - **A**ctuators: display to user, follow URL, fill in form
  - **S**ensors: HTML pages (text, graphics, scripts)

# 3. The Nature of Environments

## Specifying the Task Environment



| Agent Type | Performance Measure | Environment | Actuators | Sensors |
|---|---|---|---|---|
| Medical diagnosis system | Healthy patient, reduced costs | Patient, hospital, staff | Display of questions, tests, diagnoses, treatments | Touchscreen/voice entry of symptoms and findings |
| Satellite image analysis system | Correct categorization of objects, terrain | Orbiting satellite, downlink, weather | Display of scene categorization | High-resolution digital camera |
| Part-picking robot | Percentage of parts in correct bins | Conveyor belt with parts; bins | Jointed arm and hand | Camera, tactile and joint angle sensors |
| Refinery controller | Purity, yield, safety | Refinery, raw materials, operators | Valves, pumps, heaters, stirrers, displays | Temperature, pressure, flow, chemical sensors |
| Interactive English tutor | Student's score on test | Set of students, testing agency | Display of exercises, feedback, speech | Keyboard entry, voice |

**Figure 2.5** Examples of agent types and their PEAS descriptions.

## 3. The Nature of Environments
## 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**.
    - The sensors detect all aspects that are relevant to the choice of action; relevance, in turn, 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**.
    - One might think that in such cases the agent's plight is hopeless, but, as we discuss in Chapter 4, the agent's goals may still be achievable, sometimes with certainty.

## 3. The Nature of Environments
## Properties of Task Environments

- **Single-agent** vs. **multiagent**
  - 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, analogous to waves at the beach or leaves blowing in the wind?
  - The key distinction is whether *B*'s behavior is best described as maximizing a performance measure whose value depends on agent *A*'s behavior.
  - A **competitive** multiagent environment
    - 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.
  - A **cooperative** multiagent environment
    - In the taxi-driving environment, avoiding collisions maximizes the performance measure of all agents.

## 3. The Nature of Environments
## Properties of Task Environments

- **Deterministic** vs. **nondeterministic**
  - If the *next state* of the environment is *completely determined* by the current state and the action executed by the agent(s), then we say the environment is deterministic; otherwise, it is nondeterministic.
  - 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 nondeterministic.
  - The word **stochastic** is used by some as a synonym for "nondeterministic," but we make a distinction between the two terms;
    - We say that a model of the environment is stochastic if it explicitly deals with probabilities (e.g., "there's a 25% chance of rain tomorrow") and "nondeterministic" if the possibilities are listed without being quantified (e.g., "there's a chance of rain tomorrow").

## 3. The Nature of Environments
## Properties of Task Environments

- **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, 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.

## 3. The Nature of Environments
## Properties of Task Environments

- **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.
  - 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.

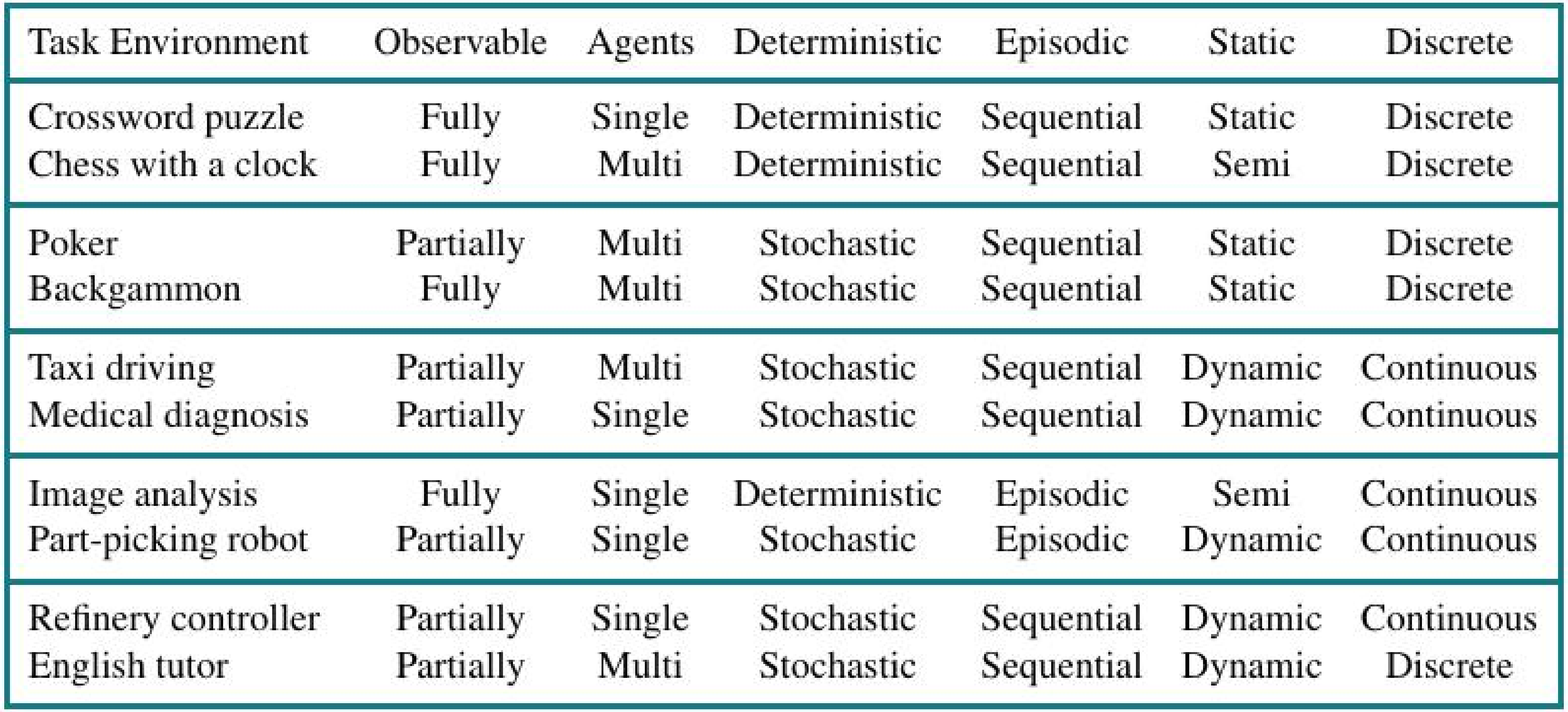
## 3. The Nature of Environments
## Properties of Task Environments

- **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.
  - The chess environment has a finite number of distinct states (excluding the clock). 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.).

## 3. The Nature of Environments
## Properties of Task Environments

- **Known** vs. **unknown**
  - Strictly speaking, this distinction refers not to the environment itself but to the agent’s (or designer’s) state of knowledge about the “laws of physics” of the environment.
  - In a **known** environment, the outcomes (or outcome  probabilities if the environment is nondeterministic) for all actions are given.
  - Obviously, if the environment is **unknown**, the agent will have to learn how it works in order to make good decisions.

## 3. The Nature of Environments
## Properties of Task Environments



| 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.

## 3. The Nature of Environments
## Properties of Task Environments

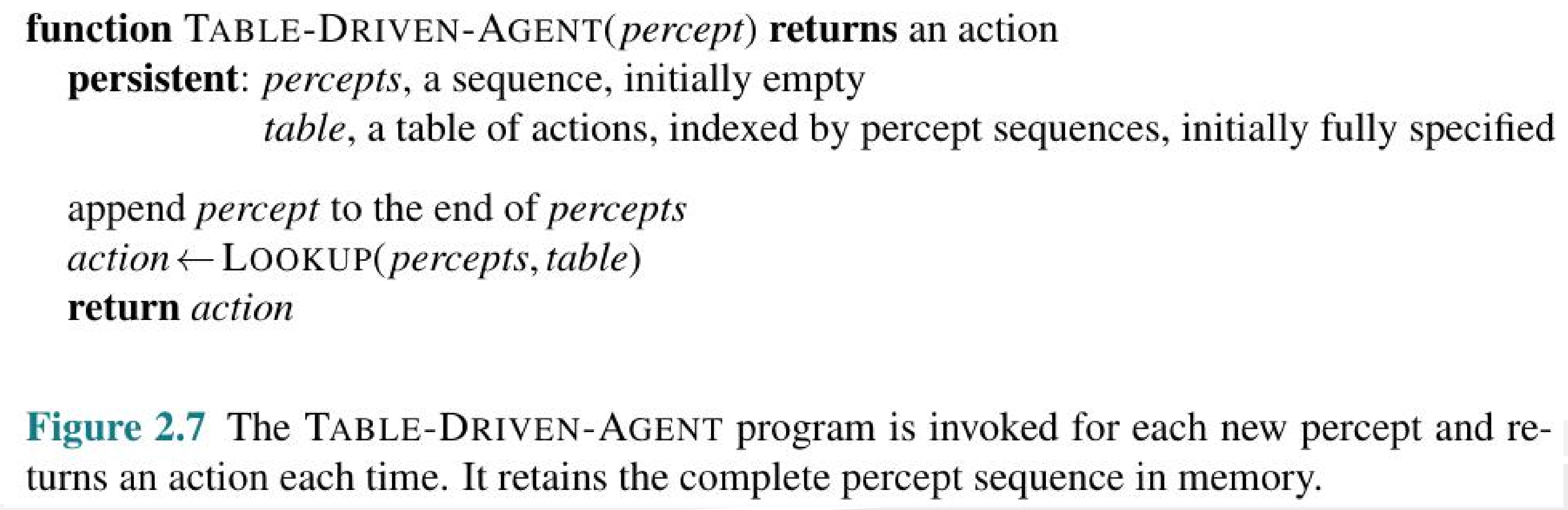
- The hardest case is partially observable, multiagent, nondeterministic, sequential, dynamic, continuous, and unknown.
- Taxi driving is hard in all these senses, except that the driver's environment is mostly known.
- Driving a rented car in a new country with unfamiliar geography, different traffic laws, and nervous passengers is a lot more exciting.

## 4. 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 **agent architecture**.
- ***agent* = *architecture* + *program***

## 4. The Structure of Agents
### Agent Programs

- The agent programs have the same skeleton:
  - They take the current percept as input from the sensors and return an action to the actuators.



```
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.

## 4. The Structure of Agents

### Agent Programs

- It is instructive to consider why the table-driven approach to agent construction is doomed to failure. !
- Let $P$ be the set of possible percepts and let $T$ be the lifetime of the agent (the total number of percepts it will receive). The lookup table will contain $\sum_{t=1}^{T} |P|^t$ entries.
  - Consider the automated taxi: the visual input from a single camera (eight cameras is typical) comes in at the rate of roughly 70 megabytes per second (30 frames per second, 1080 × 720 pixels with 24 bits of color information). This gives a lookup table with over $10^{600,000,000,000}$ entries for an hour's driving.
  - Even the lookup table for chess—a tiny, well-behaved fragment of the real world—has (it turns out) at least $10^{150}$ entries.
  - In comparison, the number of atoms in the observable universe is less than $10^{80}$.
- The daunting size of these tables means that
  - (a) no physical agent in this universe will have the space to store the table;
  - (b) the designer would not have time to create the table; and
  - (c) no agent could ever learn all the right table entries from its experience.

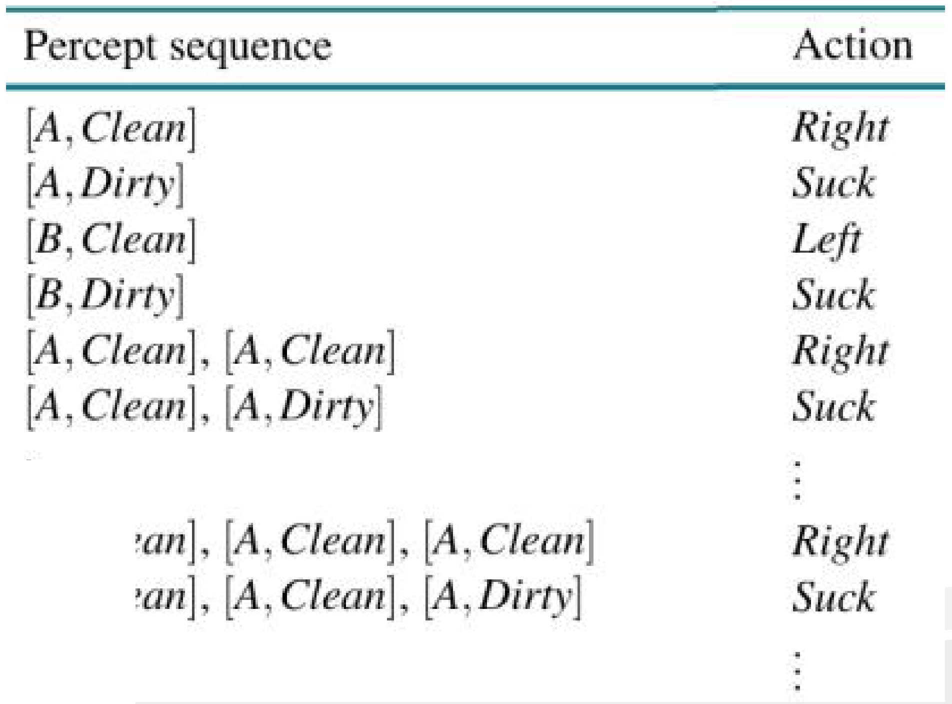
## 4. The Structure of Agents
### Agent Programs

- *The key challenge for AI is to find out how to write programs that, to the extent possible, produce rational behavior from a smallish program rather than from a vast table.*
- Four basic kinds of agent programs that embody the principles underlying almost all intelligent systems:
  - Simple reflex agents;
  - Model-based reflex agents;
  - Goal-based agents; and
  - Utility-based agents.

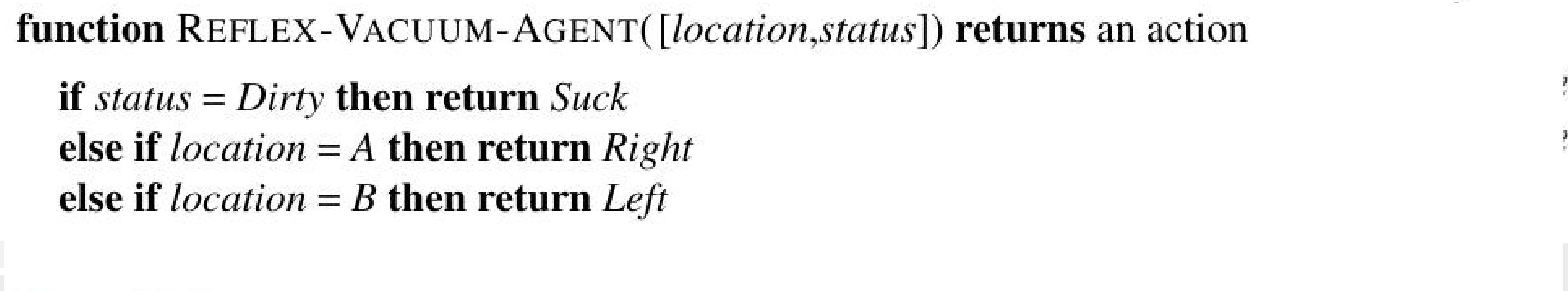
# 4. The Structure of Agents

## 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.
- The vacuum agent whose agent function is tabulated in Figure 2.3 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* |
| | ⋮ |
| an], $[A, Clean], [A, Clean]$ | *Right* |
| an], $[A, Clean], [A, Dirty]$ | *Suck* |
| | ⋮ |



```
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-location vacuum environment. This program implements the agent function tabulated in Figure 2.3.

# 4. The Structure of Agents

## Simple Reflex Agents

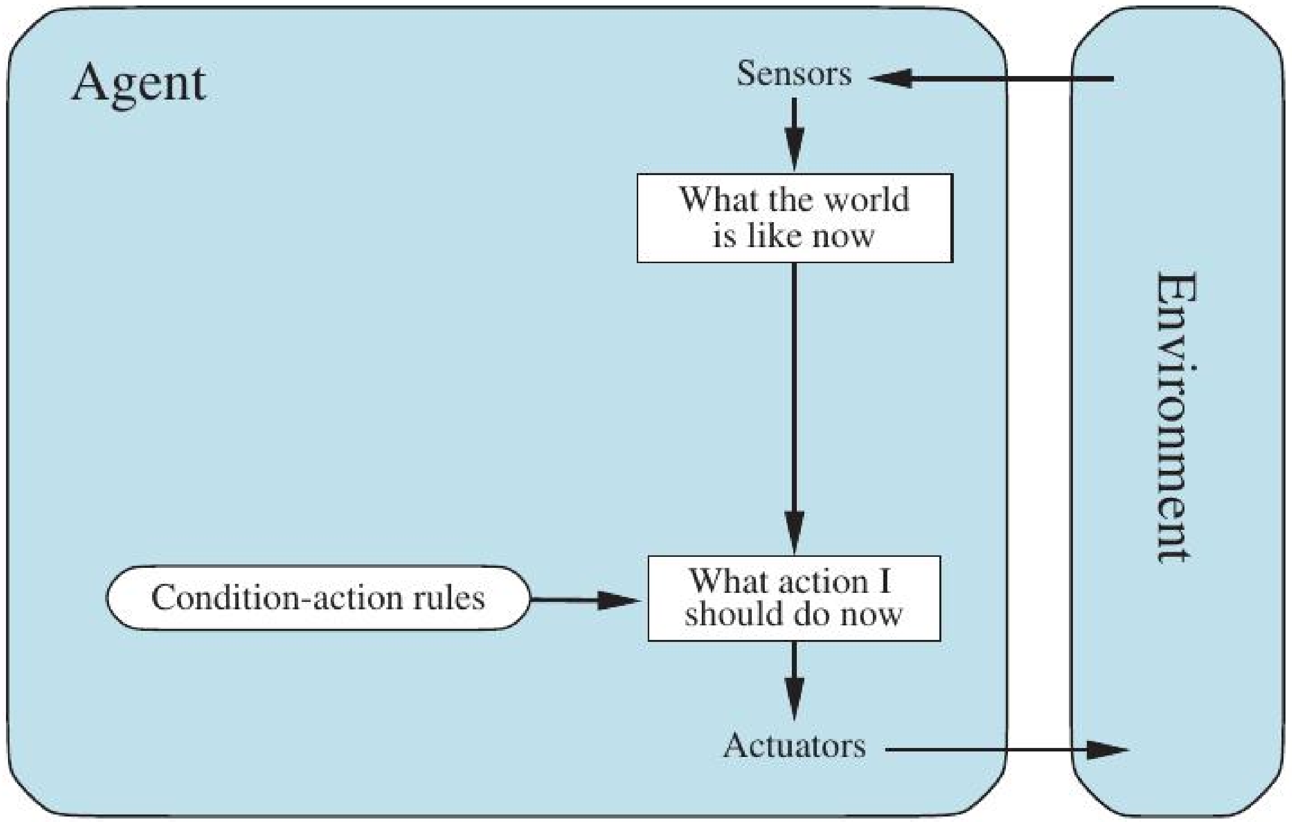
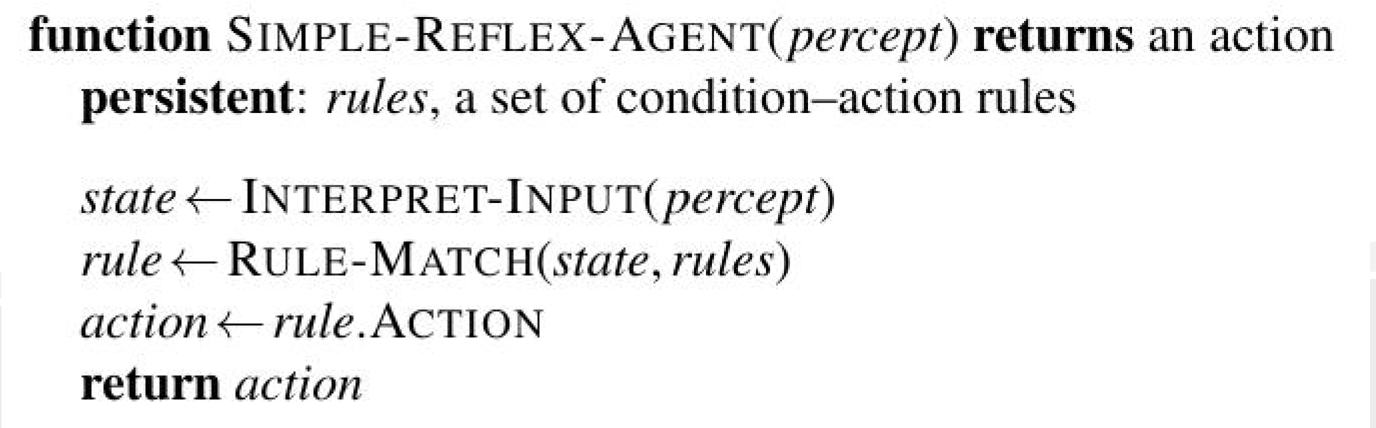
- 

**Figure 2.9** Schematic diagram of a simple reflex agent. 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.

## 4. The Structure of Agents

### Simple Reflex Agents

- Condition-action rule (situation–action rules, productions, or if–then rules)
  - 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. In other words, some processing is done on the visual input to establish the condition we call “The car in front is braking.”
    - **if** *car-in-front-is-braking* **then** *initiate-braking.*



```
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
```

- Simple reflex agents have the admirable property of being simple, but they are of limited intelligence. The agent in Figure 2.10 will work *only if the correct decision can be made on the basis of just the current percept—that is, only if the environment is fully observable.*

## 4. The Structure of Agents
### 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.*
- 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.
  - 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.

## 4. The Structure of Agents
### Model-Based Reflex Agents

- Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program in some form.
  - Transition model: we need some information about how the world changes over time, which can be divided roughly into two parts:
    - The effects of the agent's actions and
    - How the world evolves independently of the agent.
  - Sensor model: we need some information about how the state of the world is reflected in the agent's percepts.
- An agent that uses such models is called a **model-based agent**.

## 4. The Structure of Agents

### Model-Based Reflex Agents

-

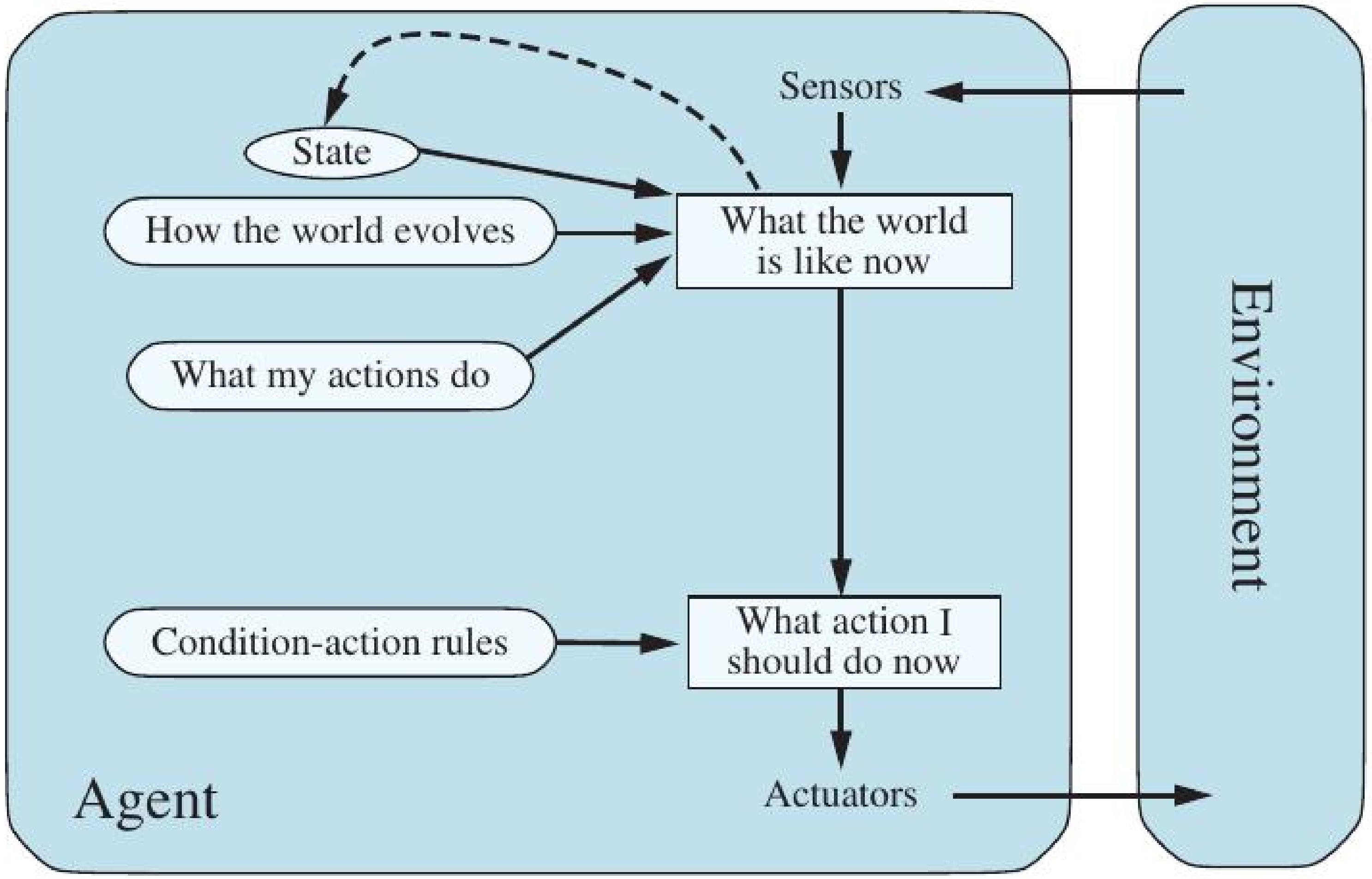
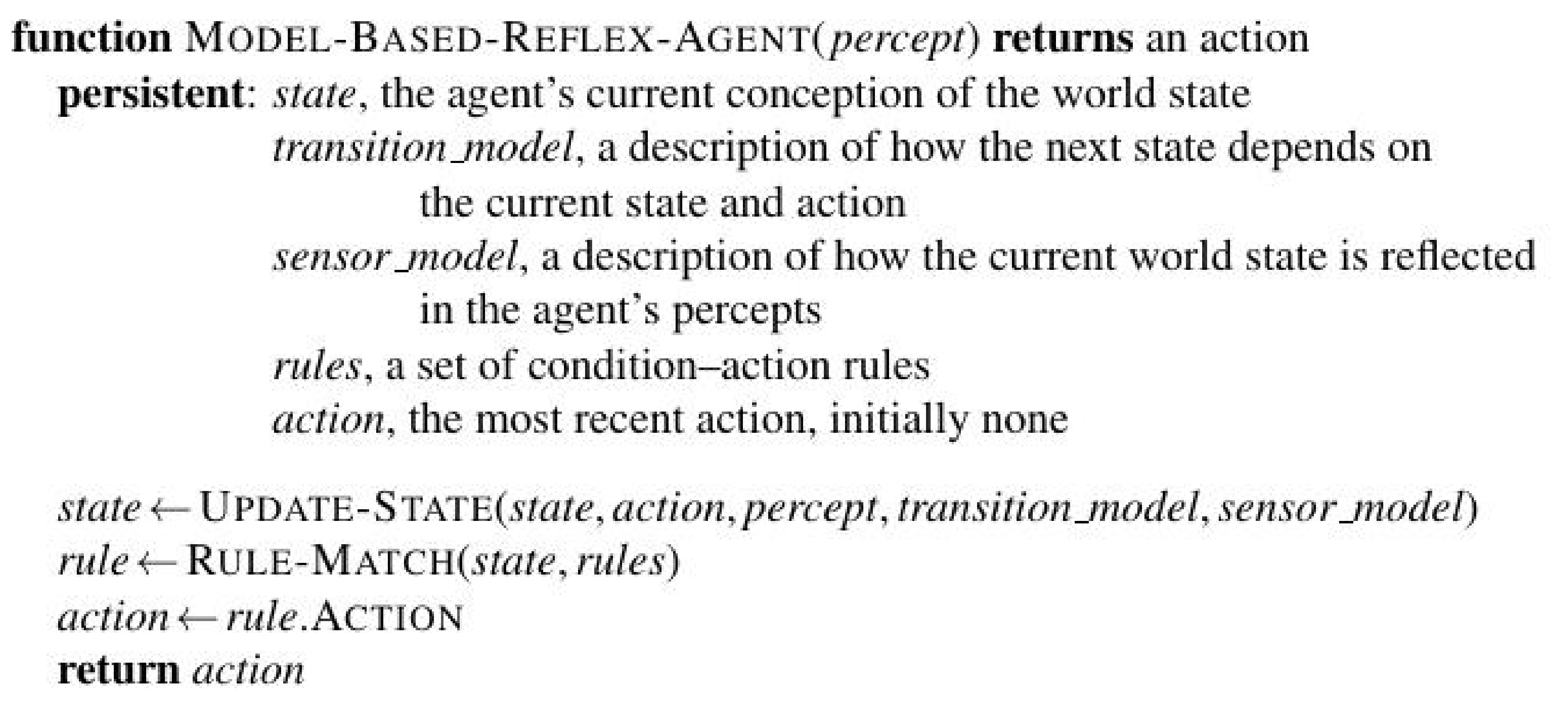


**Figure 2.11** A model-based reflex agent.

## 4. The Structure of Agents

### Model-Based Reflex Agents



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

    state ← UPDATE-STATE(state, action, percept, transition_model, sensor_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.

## 4. The Structure of Agents
### 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 agent needs some sort of **goal** information that describes situations that are desirable—for example, being at a particular 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.

# 4. The Structure of Agents

## Goal-Based Agents

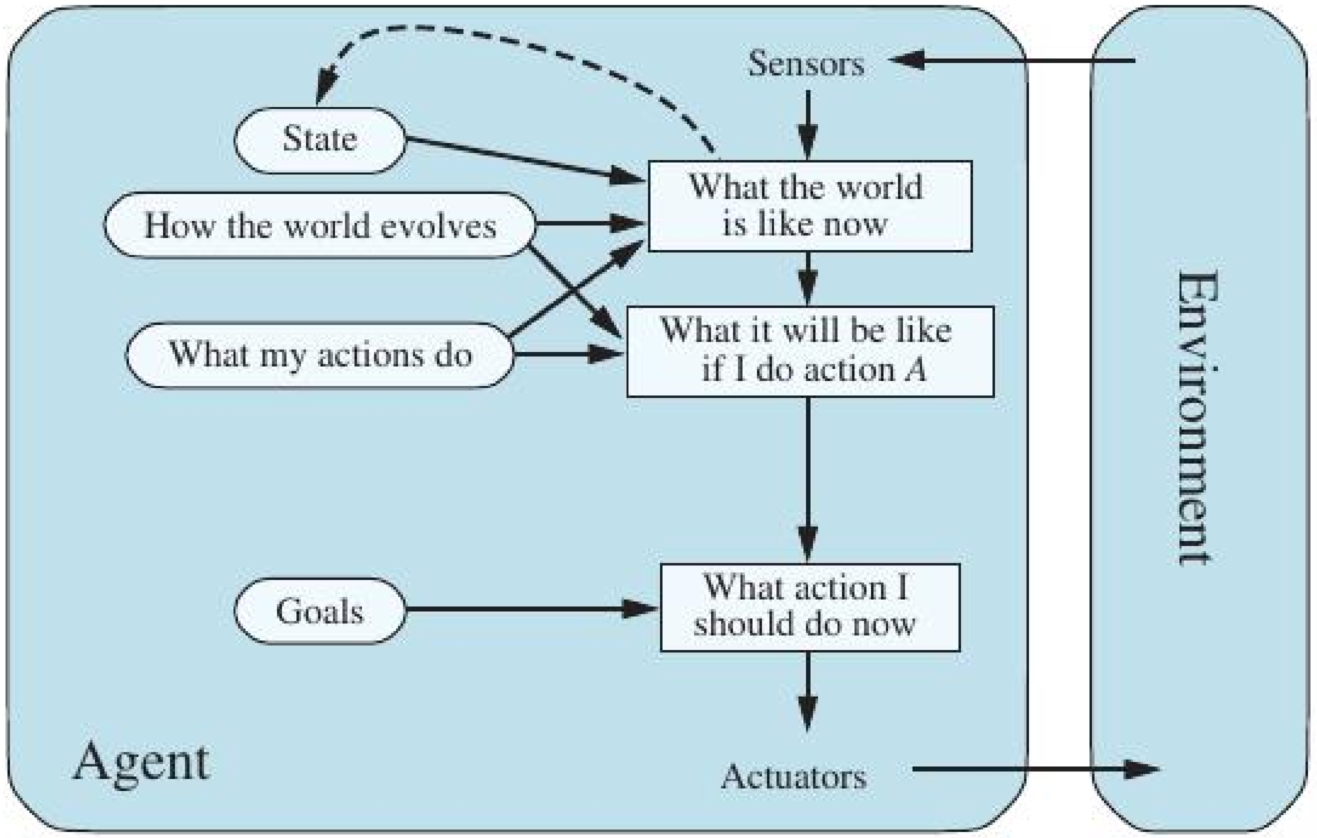

Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

**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.

## 4. The Structure of Agents

### 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.

## 4. The Structure of Agents

### Utility-Based Agents

- A performance measure assigns a score to any given sequence of environment states, so it can easily distinguish between more and less desirable ways of getting to the taxi's destination.
- An agent's **utility function** is essentially an internalization of the performance measure.
  - Provided that the internal utility function and the external performance measure are in agreement, an agent that chooses actions to maximize its utility will be *rational* according to the external performance measure.
- A utility-based agent has many advantages in terms of flexibility and learning.

# 4. The Structure of Agents

## Utility-Based Agents

- Furthermore, 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.
- Partial observability and nondeterminism are ubiquitous in the real world, and so, therefore, is decision making under uncertainty. Technically speaking, a rational utility-based agent chooses the action that maximizes the **expected utility** of the action outcomes—that is, the utility the agent expects to derive, on average, given the probabilities and utilities of each outcome.

# 4. The Structure of Agents

## Utility-Based Agents

- 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.

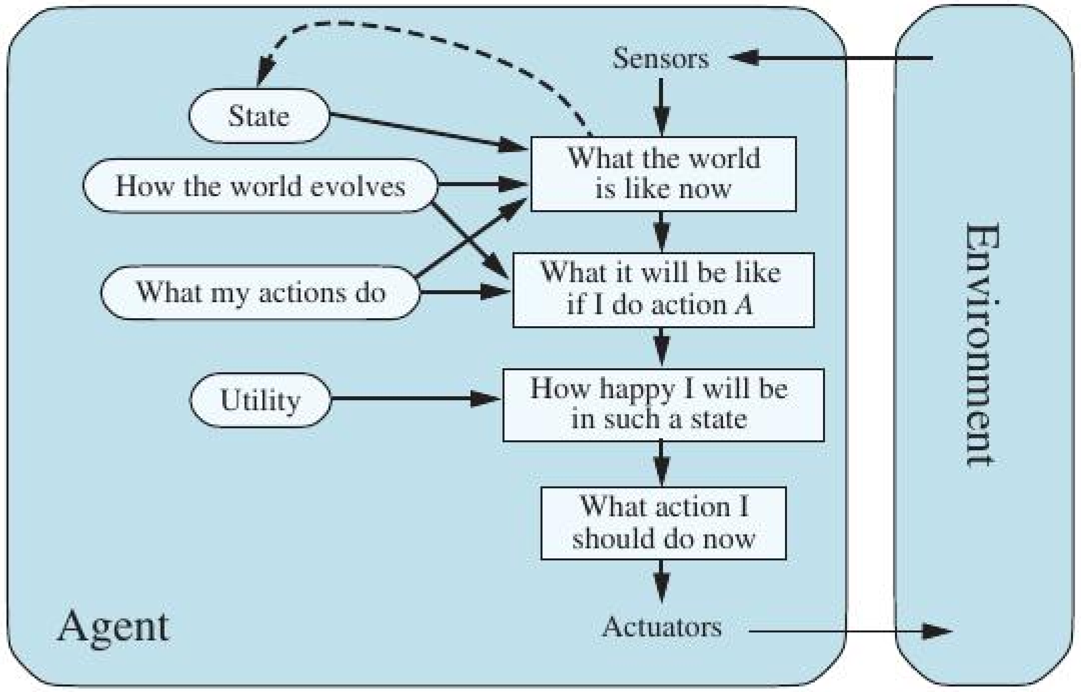

Sensors
State
What the world is like now
How the world evolves
What it will be like if I do action A
What my actions do
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

**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.