# Module 1

# Agent and Environment

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

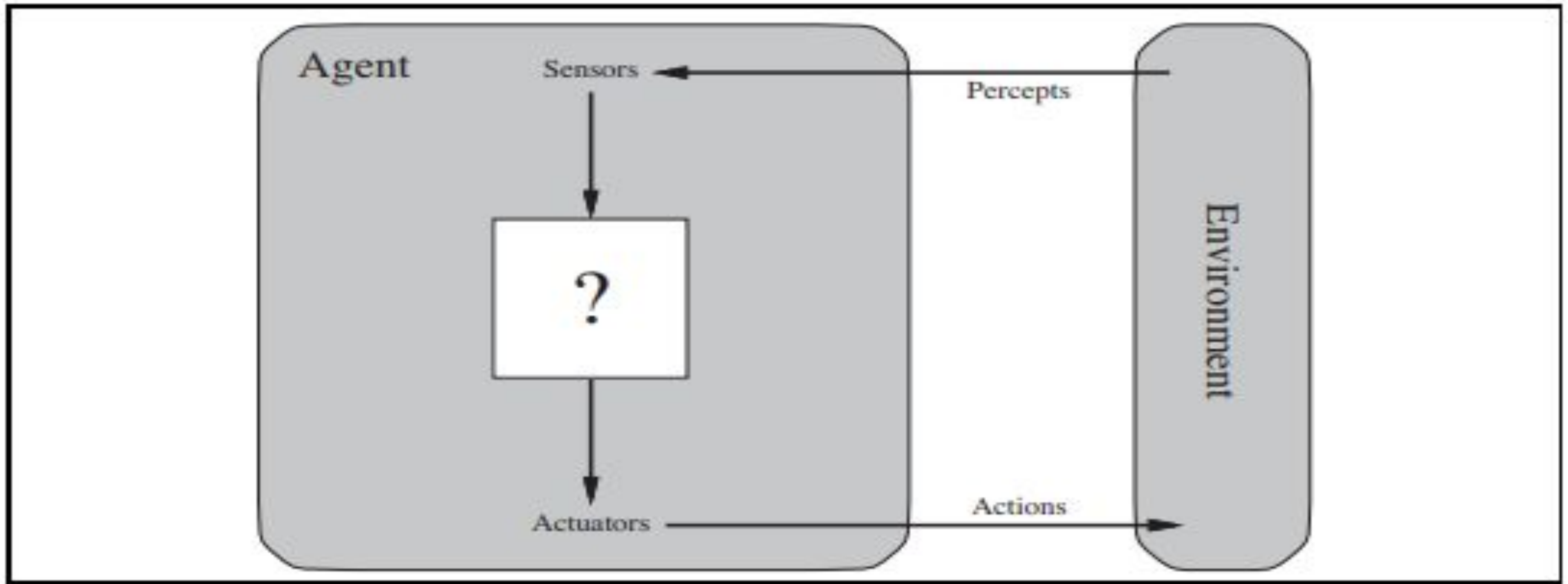


**Figure 2.1** Agents interact with environments through sensors and actuators.

- 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.

- 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.
- 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.
- Mathematically speaking, we say that 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.
- Given an agent to experiment with, we can, in principle, construct this table by trying out all possible percept sequences and recording which actions the agent does in response.
- 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.*

# Example—the vacuum-cleaner world.

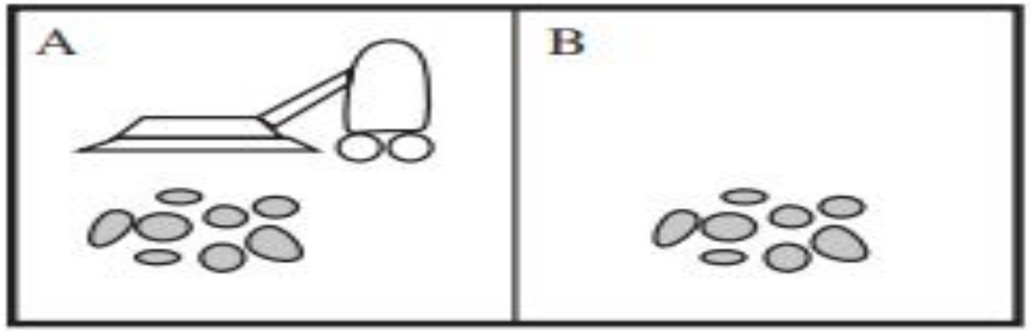
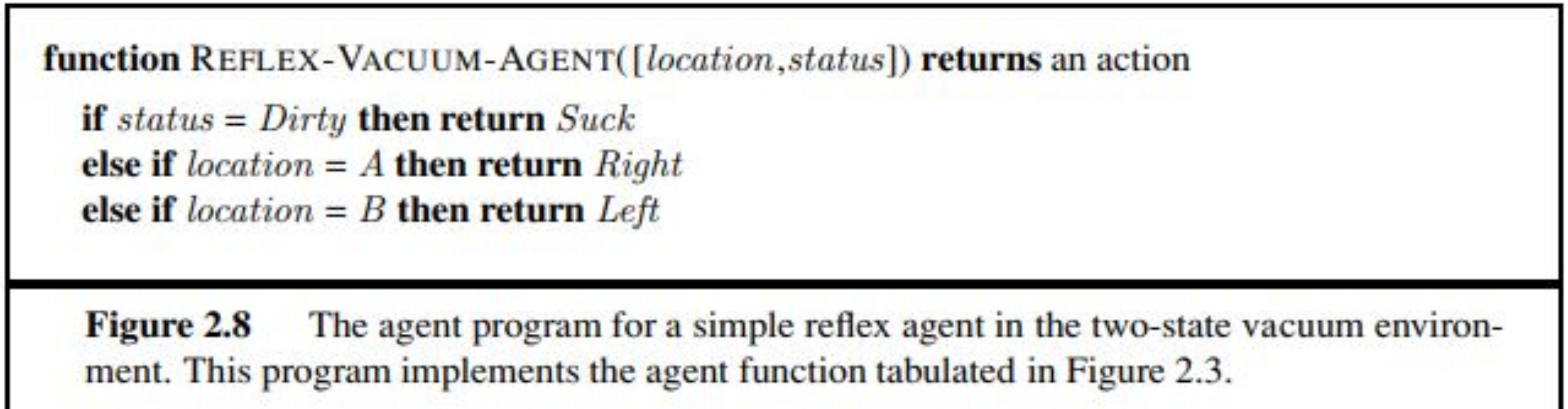


**Figure 2.2** A vacuum-cleaner world with just two locations.

| 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.

- This world is so simple that we can describe everything that happens; it’s also a made-up world, so we can invent many variations.
- 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 is 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.

## THE CONCEPT OF RATIONALITY

- A rational agent is one that does the right thing—conceptually speaking, every entry in the table for the agent function is filled out correctly.
- Obviously, doing the right thing is better than doing the wrong thing, but *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.

- Obviously, there is not one fixed performance measure for all tasks and agents; typically, a designer will devise one appropriate to the circumstances.
- Consider, for example, the vacuum-cleaner agent.
- *As a general rule, it is better to design performance measures according to what one actually wants in the environment, rather than according to how one thinks the agent should behave.*

## 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.

This leads to a 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?*
- First, we need to say what the performance measure is, what is known about the environment, and what sensors and actuators the agent has.
- 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.

- We claim that under these circumstances the agent is indeed rational; its expected performance is at least as high as any other agent's.
- 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

- We need to be careful to distinguish between rationality and omniscience.
- 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.
- If we expect an agent to do what turns out to be the best action after the fact, it will be impossible to design an agent to fulfill this specification.

- Doing actions in order to modify future percepts—sometimes called information gathering—is an important part of rationality.
- Example of information gathering is provided by the exploration that must be undertaken by a vacuum-cleaning agent in an initially unknown environment.
- A rational agent not only to gather information but also to learn as much as possible from what it perceives.
- 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.

- 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.
- After sufficient experience of its environment, the behavior of a rational agent can become effectively independent of its prior knowledge.
- Hence, the incorporation of learning allows one to design a single rational agent that will succeed in a vast variety of environments.

# THE NATURE OF ENVIRONMENTS

- Task environments- which are essentially the “problems” to which rational agents are the “solutions.”

Specifying the task environment

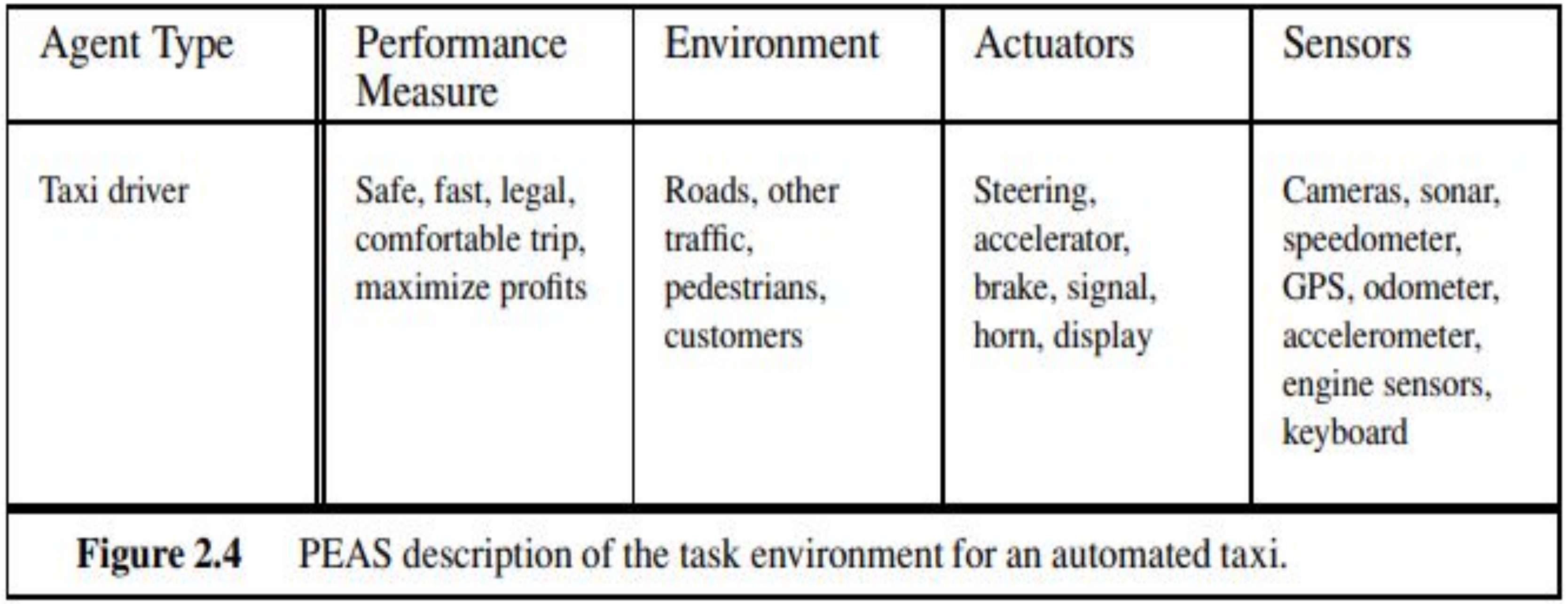
In our discussion of the rationality of the simple vacuum-cleaner agent, we had to specify the performance measure, the environment, and the agent’s actuators and sensors.

We group all these under the heading of the task environment.

For the acronymically minded, we call this the PEAS (Performance, Environment, Actuators, Sensors) description.

In designing an agent, the first step must always be to specify the task environment as fully as possible.

- Example : automated taxi driver.



| 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.

| 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.

- In contrast, some software agents (or software robots or softbots) exist in rich, unlimited domains.
- Imagine a softbot Web site operator designed to scan Internet news sources and show the interesting items to its users, while selling advertising space to generate revenue.
- To do well, that operator will need some natural language processing abilities, it will need to learn what each user and advertiser is interested in, and it will need to change its plans dynamically—for example, when the connection for one news source goes down or when a new one comes online.
- The Internet is an environment whose complexity rivals that of the physical world and whose inhabitants include many artificial and human agents.

# Properties of task environments

The range of task environments that might arise in AI is obviously vast. We can, however, identify a fairly small number of dimensions along which task environments can be categorized.

## 1. 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, 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.

## 2.Single agent vs. multiagent

- 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.
- 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, on the other hand, avoiding collisions maximizes the performance measure of all agents, so it is a partially cooperative multiagent environment.

## 3.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.
- Taxi driving is clearly stochastic in this sense, because one can never predict the behavior of traffic exactly. The vacuum world as we described it is deterministic.
- Environment is uncertain if it is not fully observable or not deterministic.

## 4. 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.

## 5. 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.
- 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.

## 6. 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 (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.).
- Input from digital cameras is discrete, strictly speaking, but is typically treated as representing continuously varying intensities and locations.

## 7. Known vs. unknown:

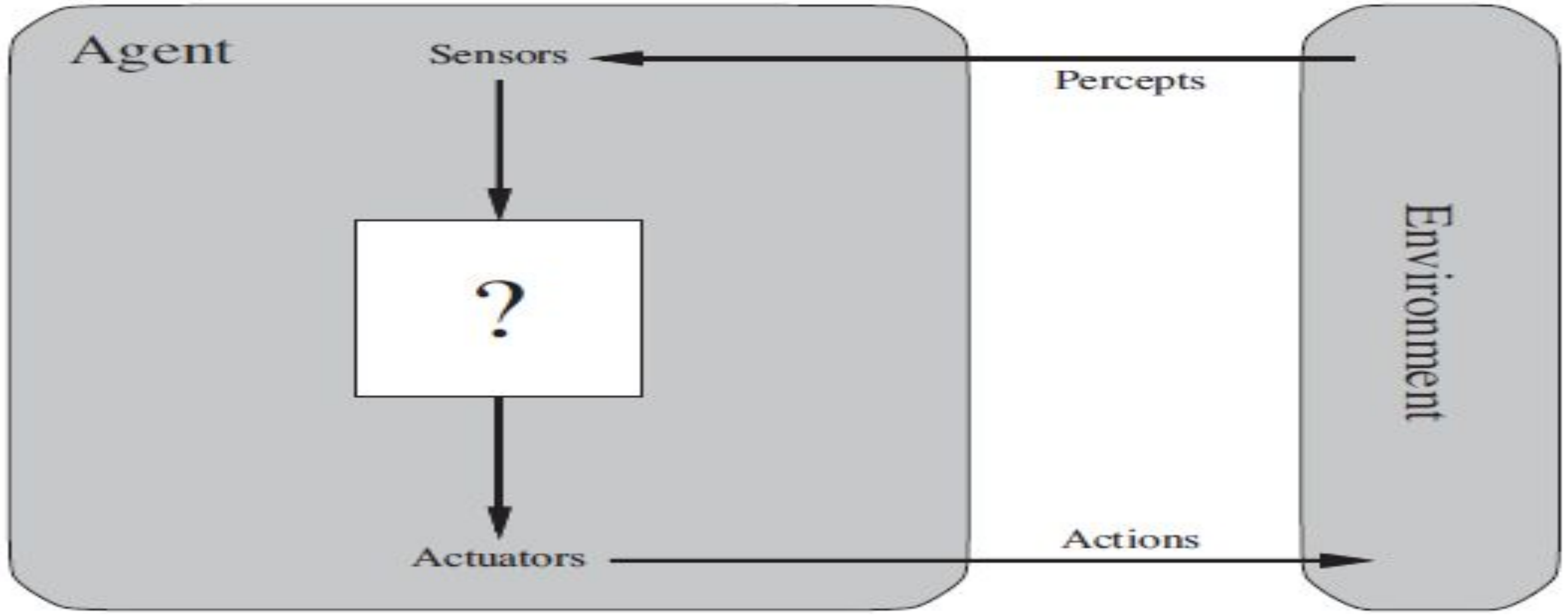
- Strictly speaking, this distinction refers not to the environment itself but to the agent’s (or designer’s) state of knowledge about the “laws of physics” of the environment.
- In a known environment, the outcomes (or outcome probabilities if the environment is stochastic) for all actions are given.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

| 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 |
| Interactive English tutor | Partially | Multi | Stochastic | Sequential | Dynamic | Discrete |

**Figure 2.6** Examples of task environments and their characteristics.

- Some experiments are often carried out not for a single environment but for many environments drawn from an environment class.
- For this reason, the code repository also includes an environment generator for each environment class that selects particular environments (with certain likelihoods) in which to run the agent.

## Agent



Agents interact with environments through sensors and actuators

## THE STRUCTURE OF AGENTS

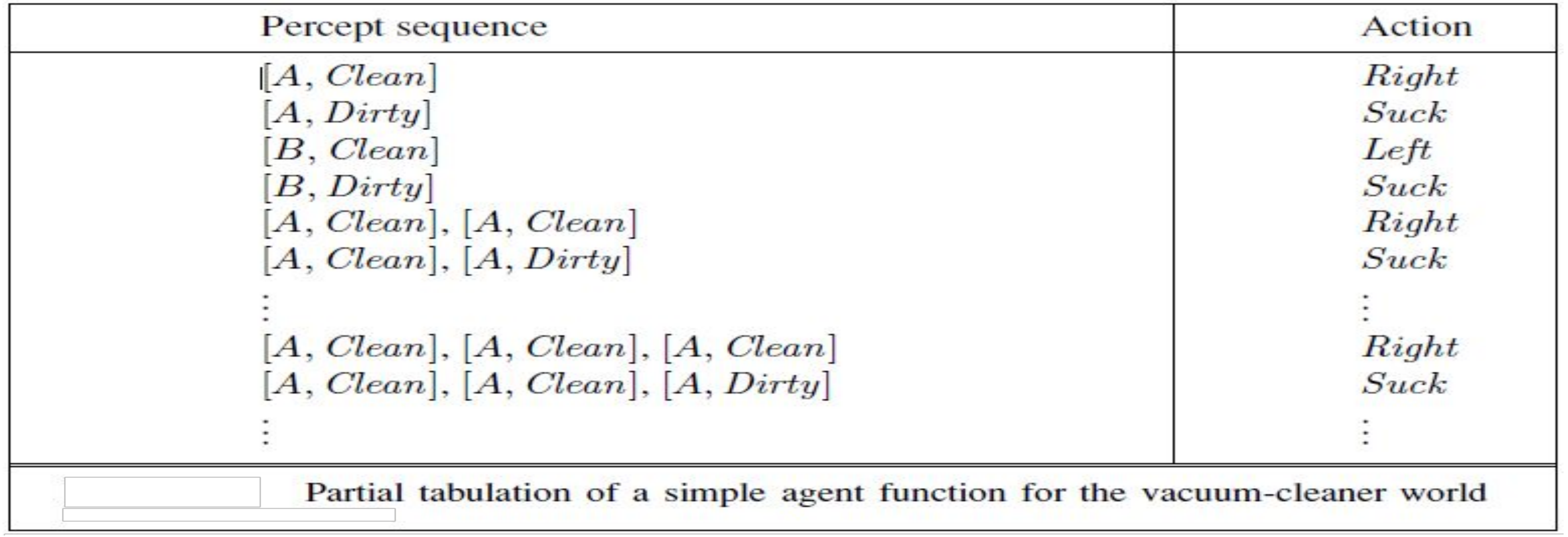
- The job of AI is to design an agent program that implements the agent function— the mapping from percepts to actions.

- 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

## Table-driven Agent

- The table represents explicitly the agent function that the agent program embodies.
- To build a rational agent in this way, the designers must construct a table that contains the appropriate action for every possible percept sequence.



| 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* |
| ⋮ | ⋮ |

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

## Table-driven Agent

- 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.

```
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
```

- 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,

  (c) no agent could ever learn all the right table entries from its experience, and

  (d) 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.

- Despite all this, TABLE-DRIVEN-AGENT *does* do what we want: it implements the desired agent function.
- 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.

## Agent programs

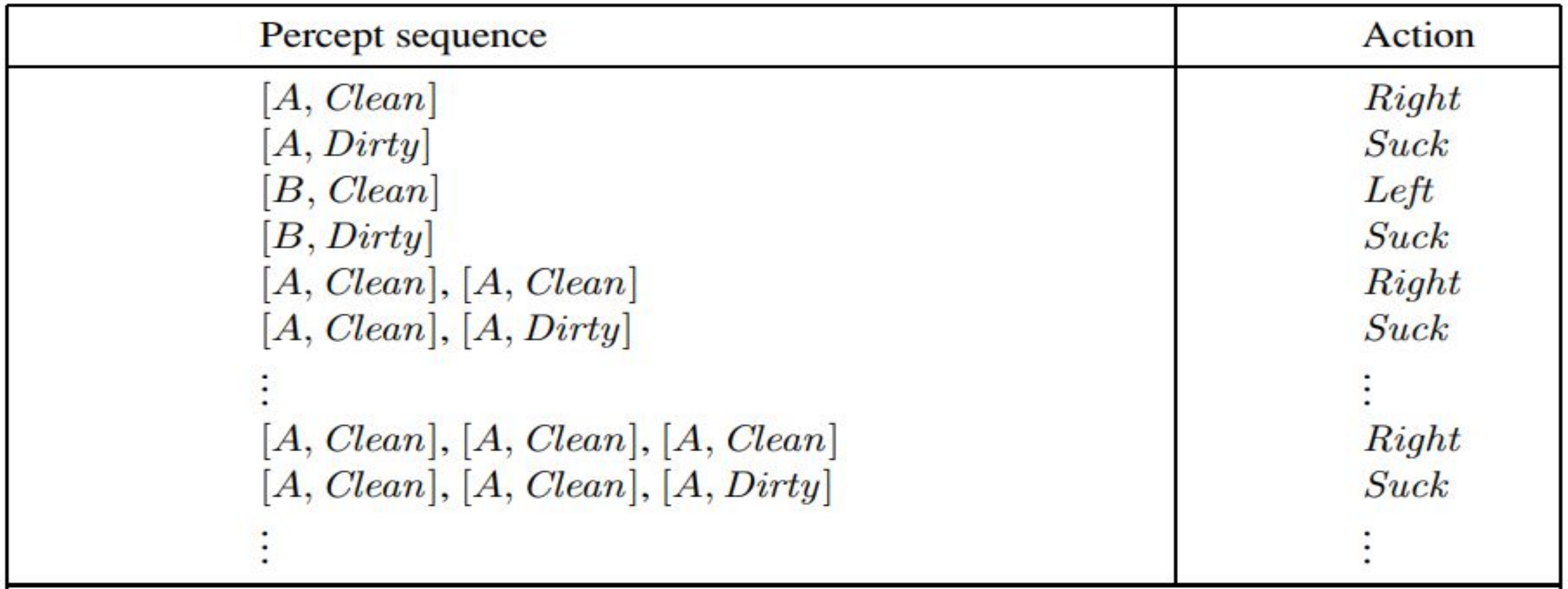
They take the current percept as input from the sensors and return an action to the actuators.

Four basic kinds of agent programs embody the principles underlying almost all intelligent systems:

- Simple reflex agents
- Model-based reflex agents
- Goal-based agents and
- Utility-based agents.

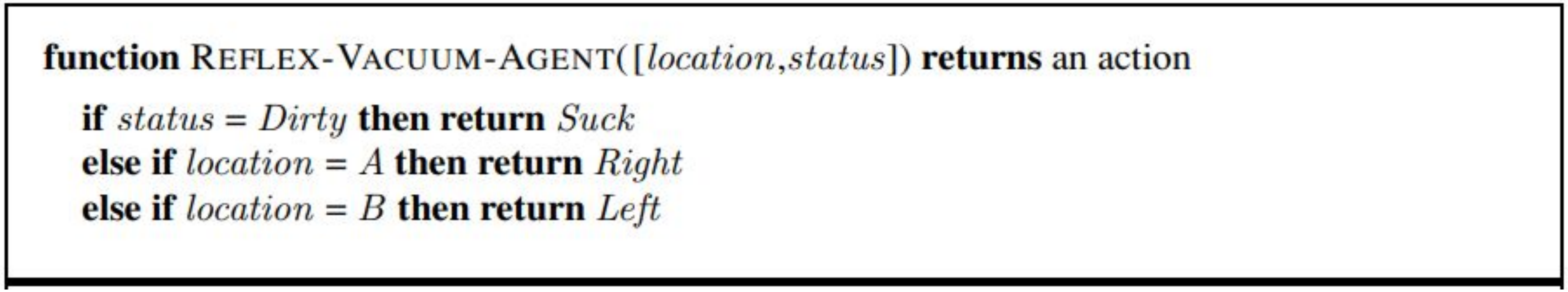
# Simple reflex agents

- 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.
- 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 |
|---|---|
| $[A, Clean]$ | $Right$ |
| $[A, Dirty]$ | $Suck$ |
| $[B, Clean]$ | $Left$ |
| $[B, Dirty]$ | $Suck$ |
| $[A, Clean], [A, Clean]$ | $Right$ |
| $[A, Clean], [A, Dirty]$ | $Suck$ |
| ⋮ | ⋮ |
| $[A, Clean], [A, Clean], [A, Clean]$ | $Right$ |
| $[A, Clean], [A, Clean], [A, Dirty]$ | $Suck$ |
| ⋮ | ⋮ |

- Agent program:



```
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.
- In other words, 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.*

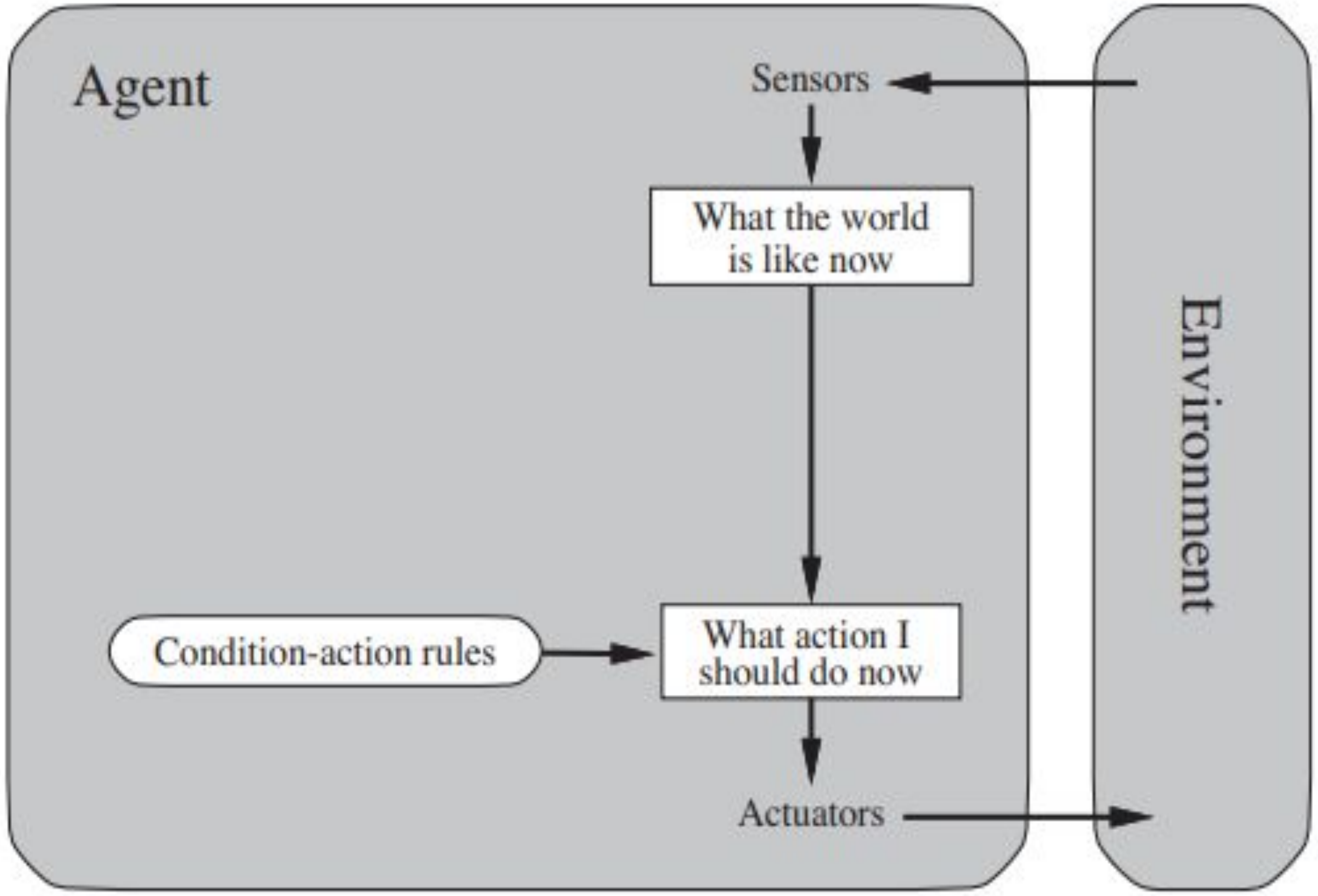
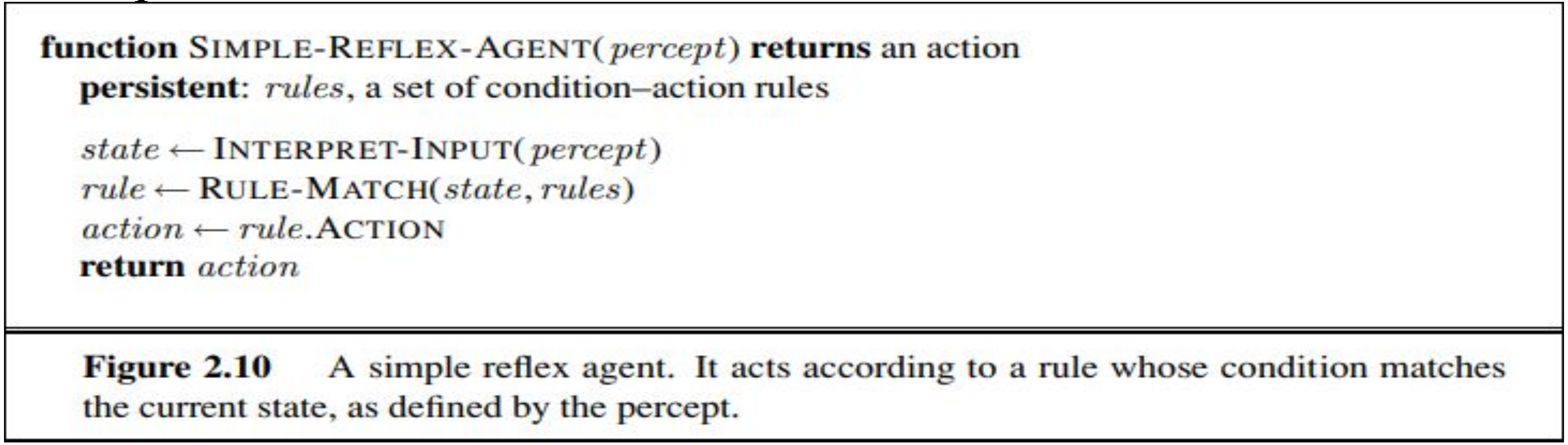


**Figure 2.9** Schematic diagram of a simple reflex agent.

Shows structure of this general program in schematic form, 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.
- 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.



```
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
```

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

```
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
```

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

- Actual implementations can be as simple as a collection of logic gates implementing a Boolean circuit.

- Simple reflex agent 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.*

- Simple reflex agents have the admirable property of being simple, but they turn out to be of limited intelligence.
- The agent in above example 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. Unfortunately, older models have different configurations of taillights, brake lights etc.
- This can create infinite loops.

- 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. Hence, a randomized simple reflex agent might outperform a deterministic simple reflex agent.

## 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.
- Example : 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.

- 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.
- 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 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.

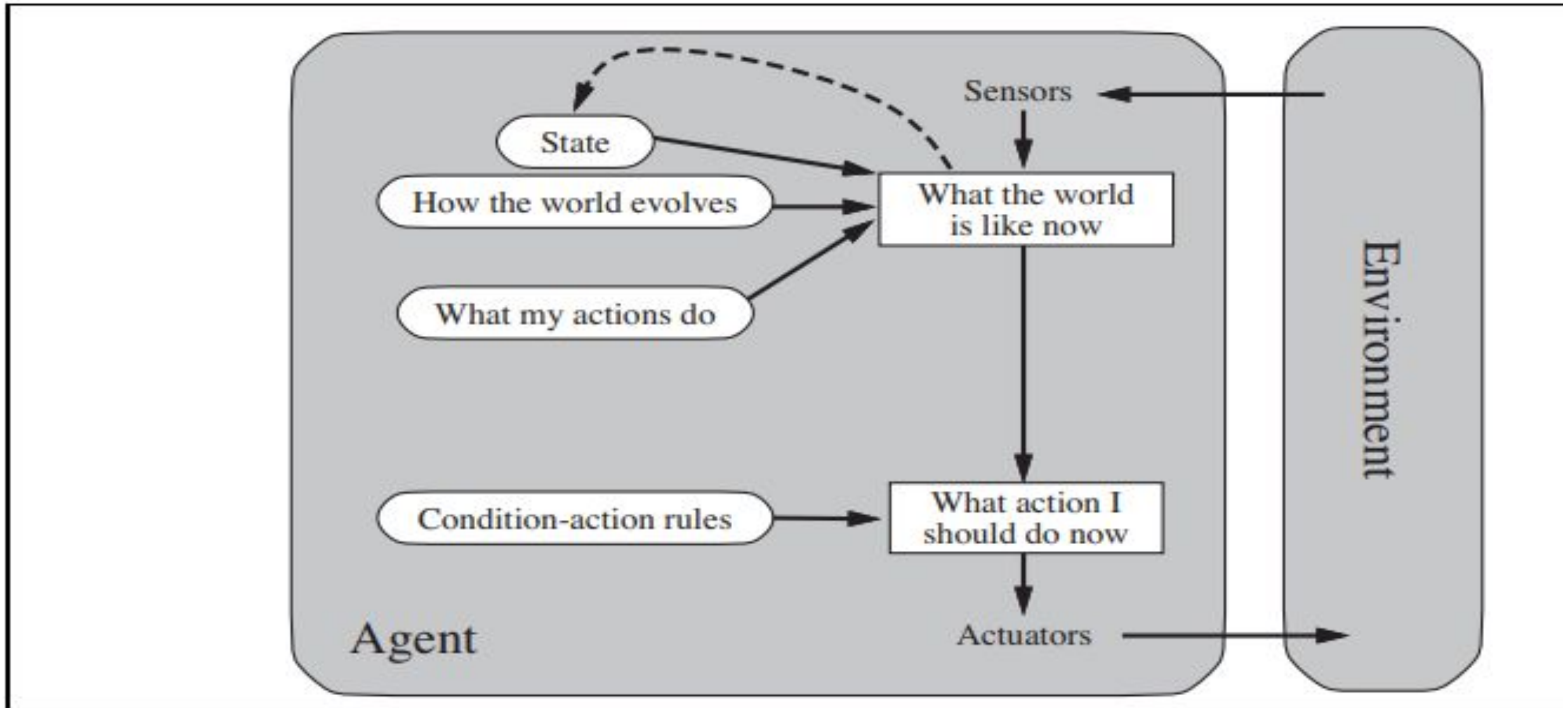


**Figure 2.11** A model-based reflex agent.

```
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.

- 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. Instead, the box labeled “what the world is like now” represents the agent’s “best guess” (or sometimes best guesses).