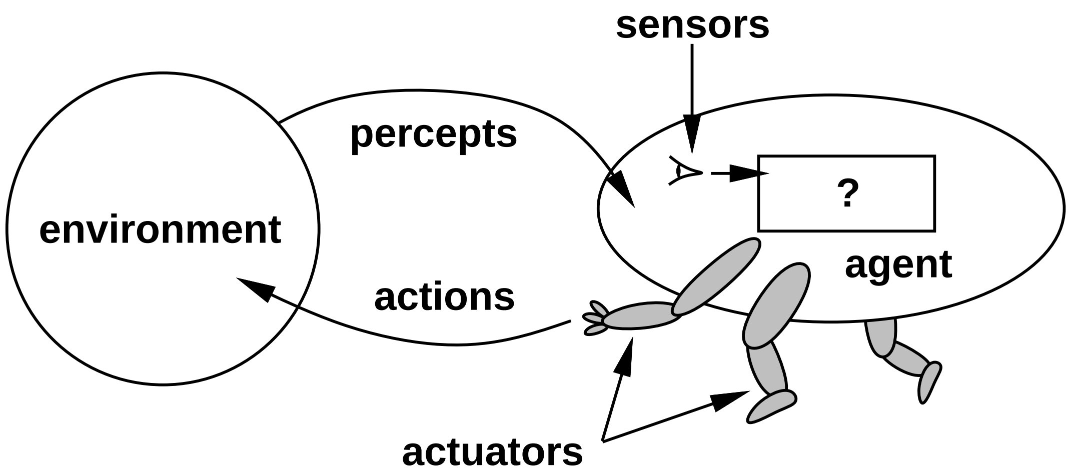
Intelligent Agents

Chapter 2

Outline

- ♢ Agents and environments
- ♢ Rationality
- ♢ PEAS (Performance measure, Environment, Actuators, Sensors)
- ♢ Environment types
- ♢ Agent types

Agents and environments



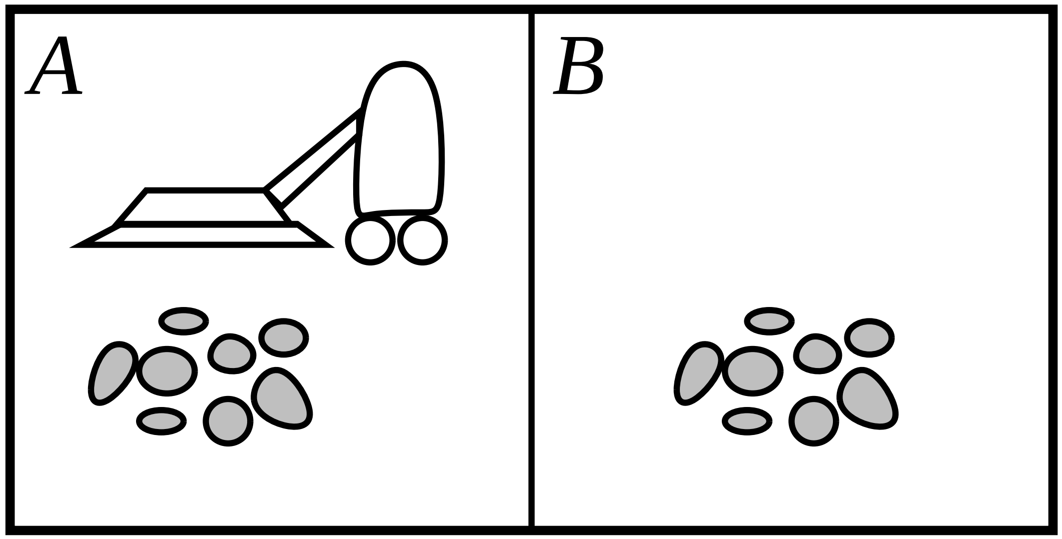
Agents include humans, robots, softbots, thermostats, etc.

The agent function maps from percept histories to actions:

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

The agent program runs on the physical architecture to produce f

Vacuum-cleaner world



Percepts: location and contents, e.g., $[A, Dirty]$

Actions: $Left$, $Right$, $Suck$, $NoOp$

A vacuum-cleaner agent

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

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*

What is the **right** function?
Can it be implemented in a small agent program?

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?

A rational agent chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date

Rational $\neq$ omniscient
- percepts may not supply all relevant information

Rational $\neq$ clairvoyant
- action outcomes may not be as expected

Hence, rational $\neq$ successful

Rational $\Rightarrow$ exploration, learning, autonomy

PEAS

To design a rational agent, we must specify the task environment

Consider, e.g., the task of designing an automated taxi:

Performance measure??

Environment??

Actuators??

Sensors??

PEAS

To design a rational agent, we must specify the task environment

Consider, e.g., the task of designing an automated taxi:

Performance measure?? safety, destination, profits, legality, comfort, . . .

Environment?? US streets/freeways, traffic, pedestrians, weather, . . .

Actuators?? steering, accelerator, brake, horn, speaker/display, . . .

Sensors?? video, accelerometers, gauges, engine sensors, keyboard, GPS, . . .

Internet shopping agent

Performance measure??

Environment??

Actuators??

Sensors??

Internet shopping agent

Performance measure?? price, quality, appropriateness, efficiency

Environment?? current and future WWW sites, vendors, shippers

Actuators?? display to user, follow URL, fill in form

Sensors?? HTML pages (text, graphics, scripts)

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??				
Deterministic??				
Episodic??				
Static??				
Discrete??				
Single-agent??				

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??				
Episodic??				
Static??				
Discrete??				
Single-agent??				

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**. 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.

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??				
Static??				
Discrete??				
Single-agent??				

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. (In our definition, we ignore uncertainty that arises purely from the actions of other agents in a multiagent environment; thus, a game can be deterministic even though each agent may be unable to predict the actions of the others.) 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 (Exercise 2.13). We say an environment is **uncertain** if it is not fully observable or not deterministic. One final note: 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. Nondeterministic environment descriptions are usually associated with performance measures that require the agent to succeed for *all possible* outcomes of its actions.

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??				
Discrete??				
Single-agent??				

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.3 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.

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??	Yes	Semi	Semi	No
Discrete??				
Single-agent??				

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.

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??	Yes	Semi	Semi	No
Discrete??	Yes	Yes	Yes	No
Single-agent??				

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.

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??	Yes	Semi	Semi	No
Discrete??	Yes	Yes	Yes	No
Single-agent??	Yes	No	Yes (except auctions)	No

The environment type largely determines the agent design

The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent

Single agent vs. **multiagent**: The distinction between single-agent and multiagent 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. There are, however, some subtle issues. 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, 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. 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 COMPETITIVE 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. 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; for example, **communication** often emerges as a rational behavior in multiagent environments; in some competitive environments, **randomized behavior** is rational because it avoids the pitfalls of predictability.

Agent types

Four basic types in order of increasing generality:
– simple reflex agents
– reflex agents with state or model base reflex agent
– goal-based agents
– utility-based agents

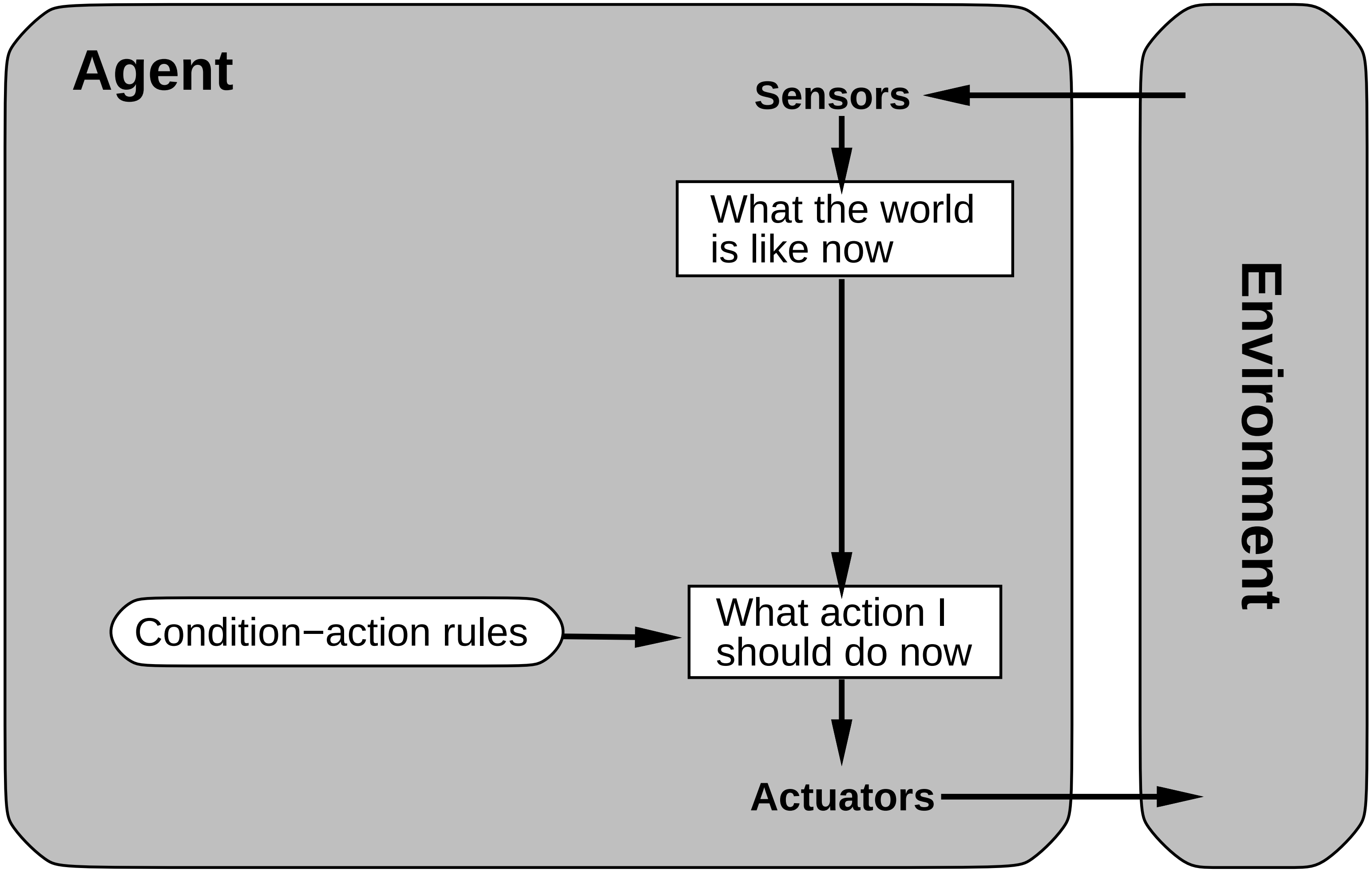
All these can be turned into learning agents

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. In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program's action choices to the actuators as they are generated.

Simple reflex agents



Example

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*

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.

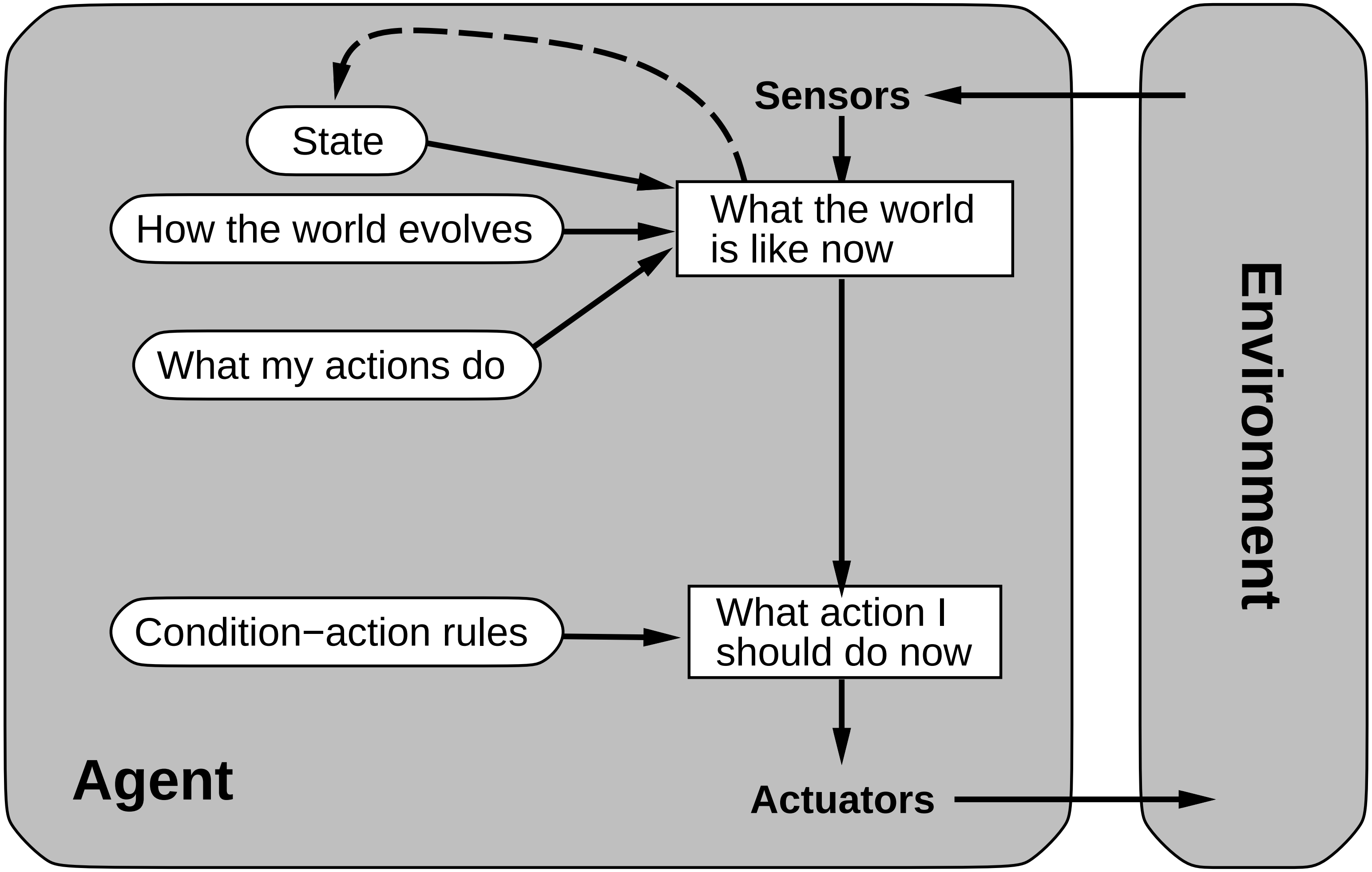
)

- 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 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.

- 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 RANDOMIZATION 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.

Reflex agents with state



Example

function REFLEX-VACUUM-AGENT([*location*,*status*]) **returns** an action
static: $last\_A$, $last\_B$, numbers, initially ∞

  if $status = Dirty$ **then** . . .

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*

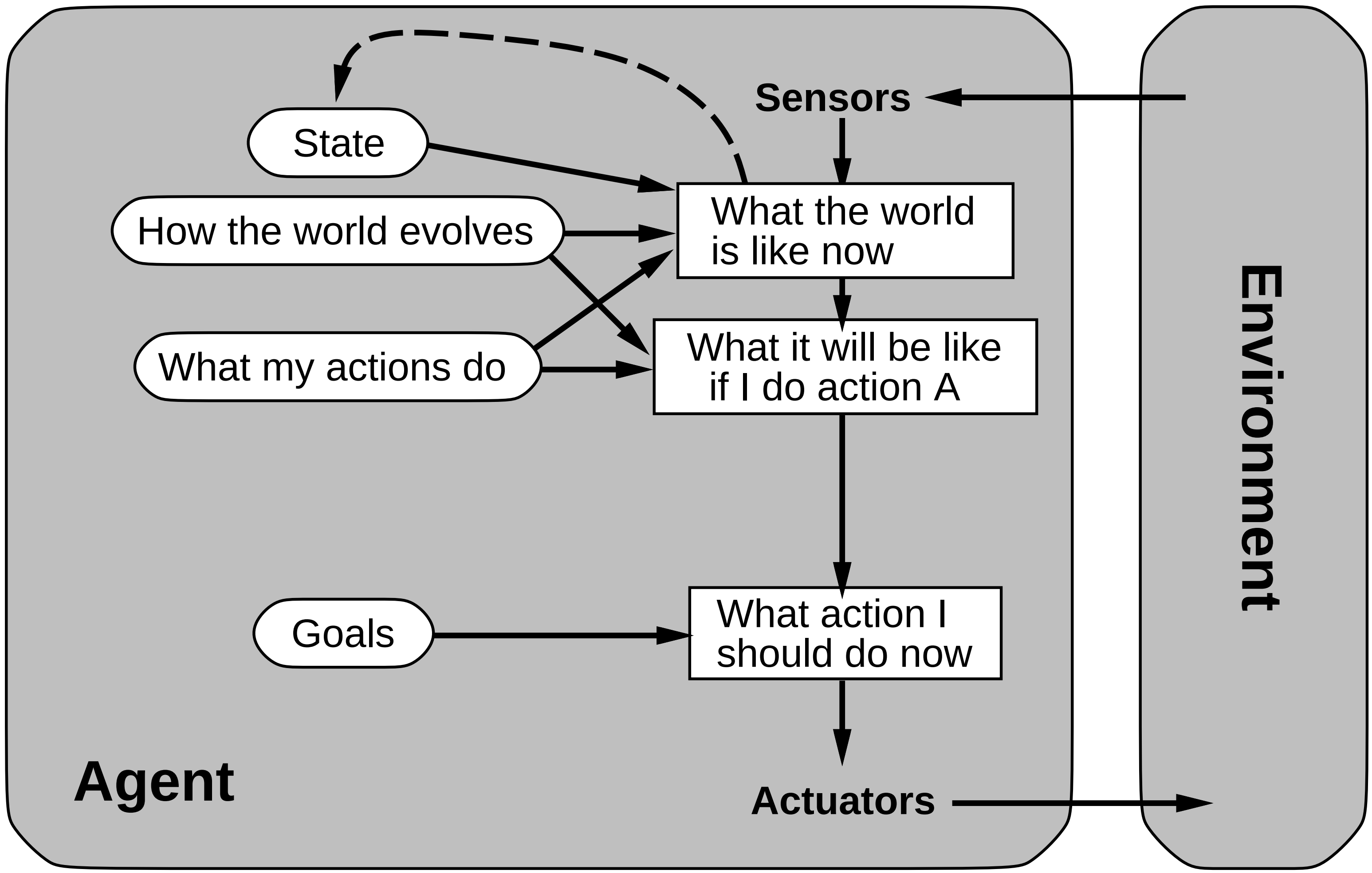
Figure 2.12 A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

- 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.

- 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. 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.

- 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" (Figure 2.11) 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

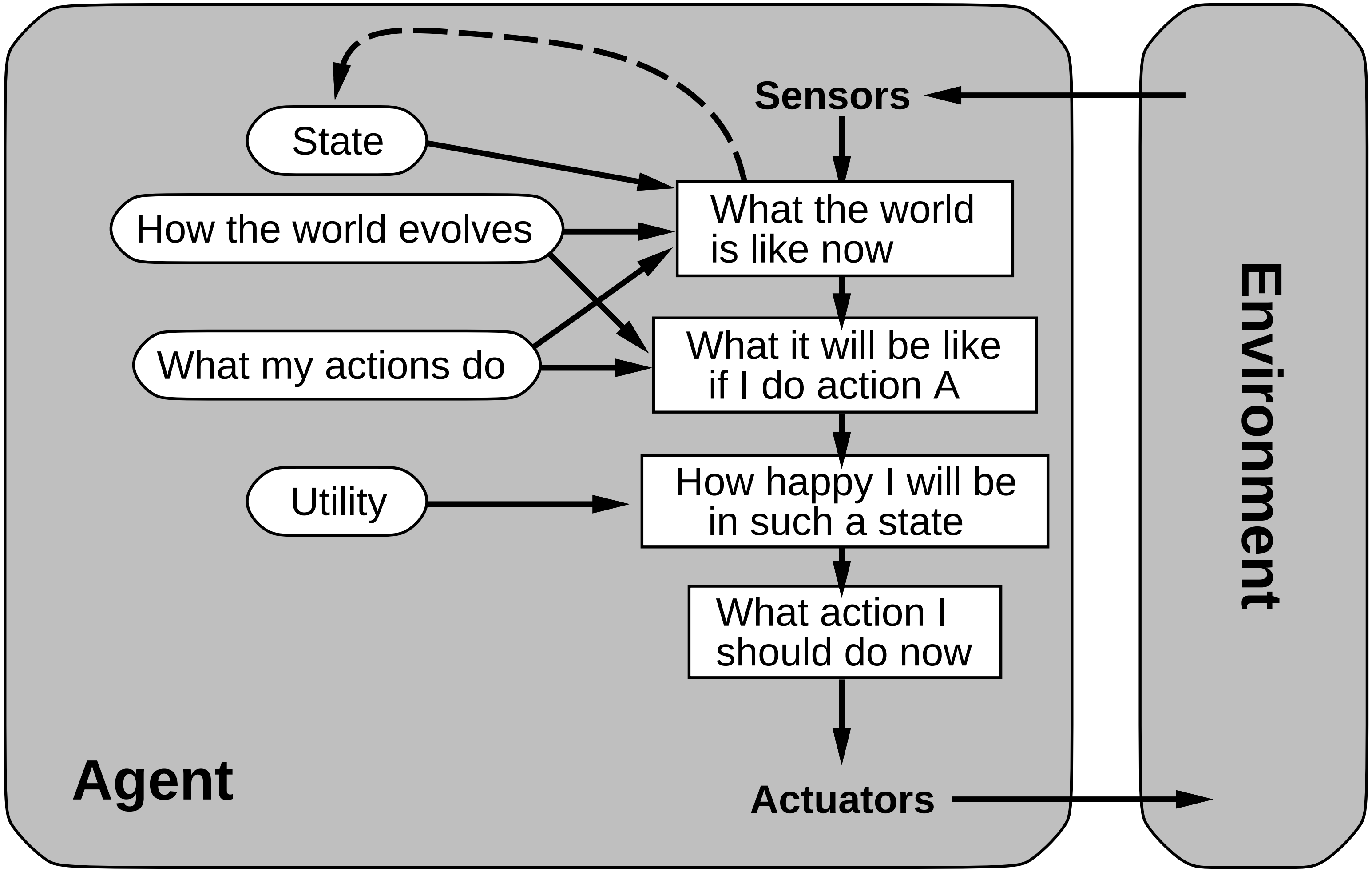


- Knowing something about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to. In other words, as well as a current state description, the 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.

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

- Although the goal-based agent appears less efficient, it is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified. If it starts to rain, the agent can update its knowledge of how effectively its brakes will operate; this will automatically cause all of the relevant behaviors to be altered to suit the new conditions. For the reflex agent, on the other hand, we would have to rewrite many condition–action rules. 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

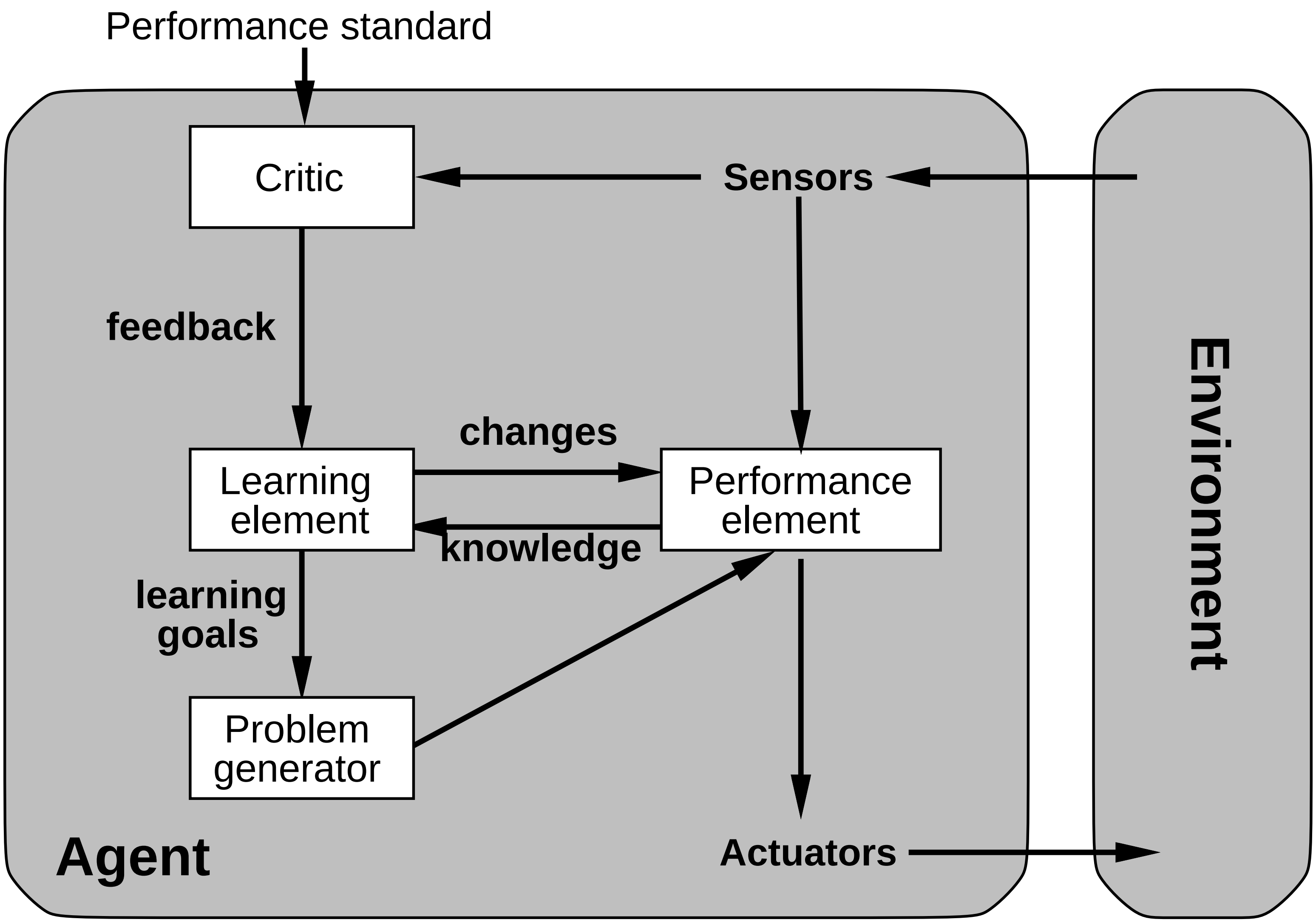


- Goals alone are not enough to generate high-quality behavior in most environments. For example, many action sequences will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others. Goals just provide a crude binary distinction between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

- 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.

- Partial observability and stochasticity 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. Rational agent must behave as if it possesses a utility function whose expected value it tries to maximize. An agent that possesses an explicit utility function can make rational decisions with a general-purpose algorithm that does not depend on the specific utility function being maximized. In this way, the “global” definition of rationality—designating as rational those agent functions that have the highest performance—is turned into a “local” constraint on rational-agent designs that can be expressed in a simple program.

Learning agents



- A learning agent can be divided into four conceptual components, as shown in Figure (previous slide).

- 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.

- The design of the learning element depends very much on the design of the performance element. 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?" Given an agent design, learning mechanisms can be constructed to improve every part of the agent.

- The critic tells the learning element how well the agent is doing with respect to a fixed performance standard. The critic is necessary because the percepts themselves provide no indication of the agent's success. For example, a chess program could receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; the percept itself does not say so.

- It is important that the performance standard be fixed. Conceptually, one should think of it as being outside the agent altogether because the agent must not modify it to fit its own behavior.

- 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. The point is that if the performance element had its way, it would keep doing the actions that are best, given what it knows. But if the agent is willing to explore a little and do some perhaps suboptimal actions in the short run, it might discover much better actions for the long run. The problem generator's job is to suggest these exploratory actions.

Summary

Agents interact with environments through actuators and sensors

The agent function describes what the agent does in all circumstances

The performance measure evaluates the environment sequence

A perfectly rational agent maximizes expected performance

Agent programs implement (some) agent functions

PEAS descriptions define task environments

Environments are categorized along several dimensions:
observable? deterministic? episodic? static? discrete? single-agent?

Several basic agent architectures exist:
reflex, reflex with state, goal-based, utility-based