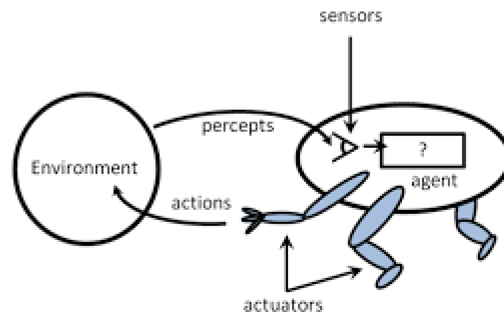
COE 4213564

Introduction to Artificial Intelligence

Intelligent Agents

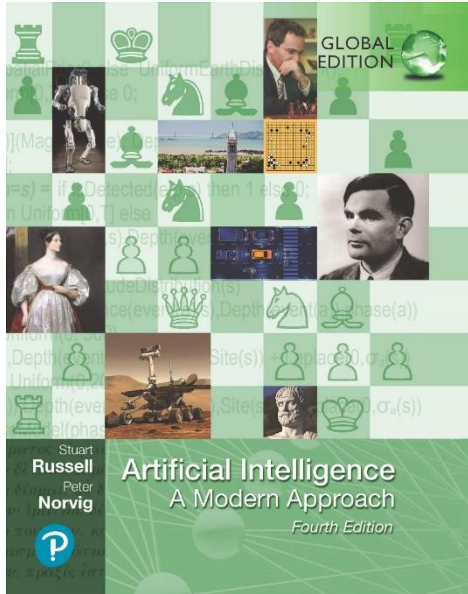
Chapter 2



Spring 2022

Many slides are adapted from CS 188 (http://ai.berkeley.edu), CIS 521, CS 221.

Fourth Edition, Global Edition



Chapter 2

Intelligent Agents

Chapter 2 © 2022 Pearson Education Ltd.

Pearson

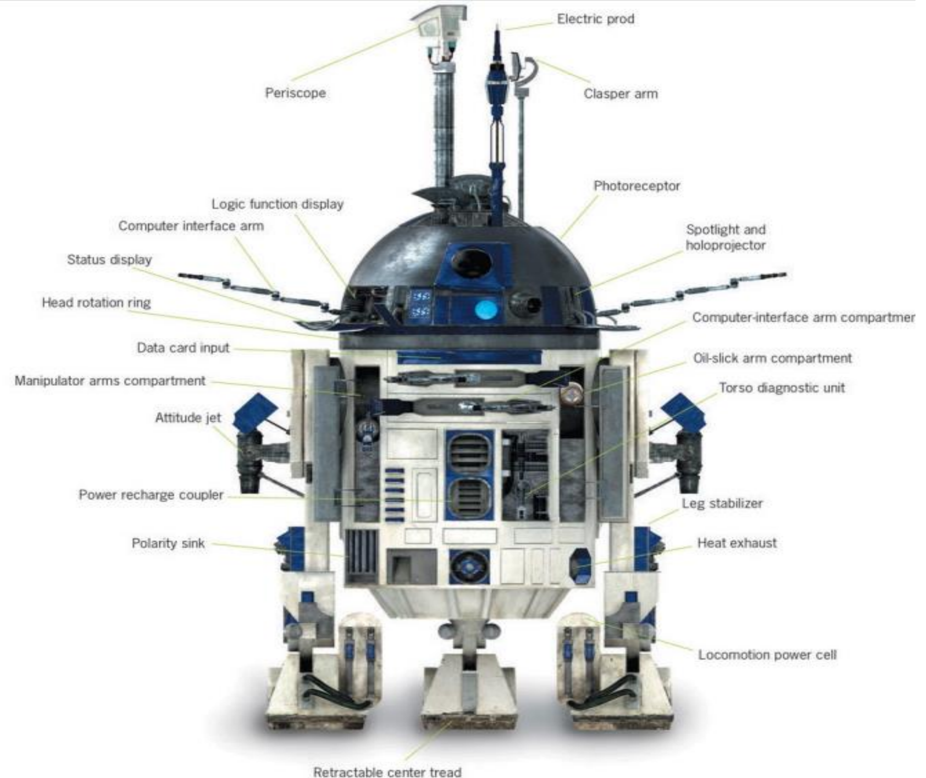
Pearson

Agents

Agents

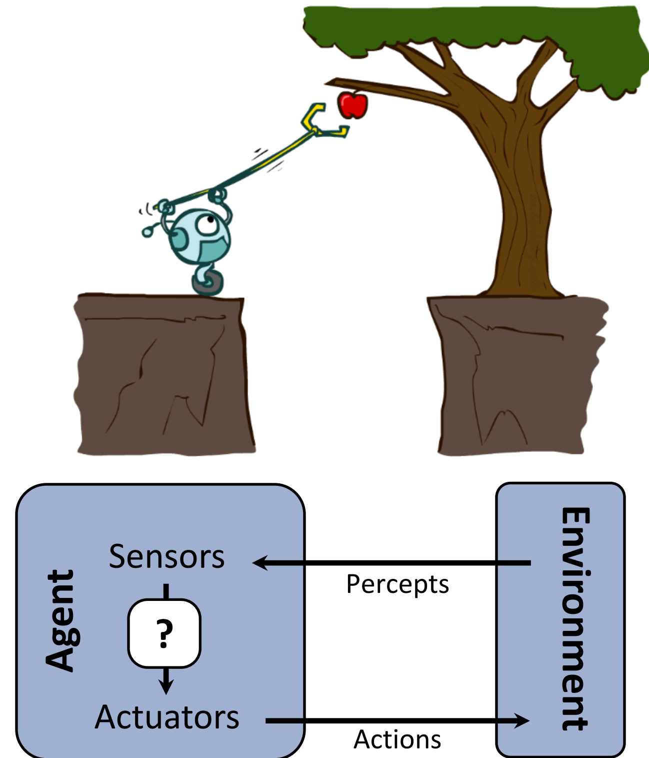
An **agent** is anything that **perceives** its environment through **sensors** and can **act** on its environment through **actuators**

A **percept** is the agent's perceptual inputs at any given instance.

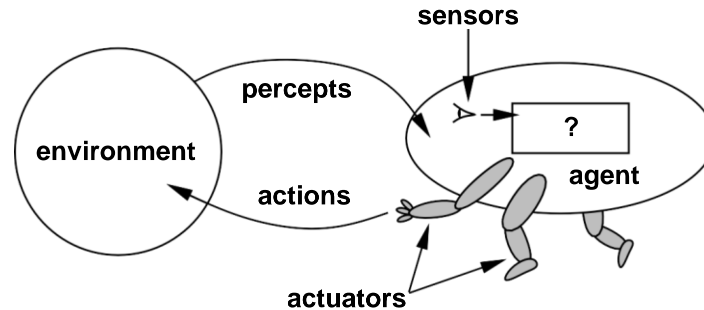


Designing Rational Agents

- An **agent** is an entity that *perceives* and *acts*.
- A **rational agent** selects actions that maximize its (expected) **utility**.
- Characteristics of the **percepts, environment,** and **action space** dictate techniques for selecting rational actions
- **This course** is about:
  - General AI techniques for a variety of problem types
  - Learning to recognize when and how a new problem can be solved with an existing technique



Agents and environments



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

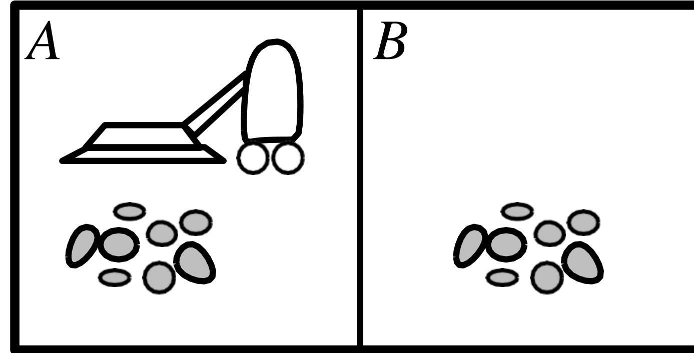
An agent can be anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators

The agent function maps from percept histories to actions:

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

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*
.	.

```
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 $/=$ omniscient

– percepts may not supply all relevant information

Rational $/=$ clairvoyant

– action outcomes may not be as expected

Hence, rational $/=$ successful

Rational $\Rightarrow$ exploration, learning, autonomy

© 2022 Pearson Education Ltd.

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??

Pearson

© 2022 Pearson Education Ltd.

PEAS

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

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

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.

Internet shopping agent

Performance measure??

Environment??

Actuators??

Sensors??

© 2022 Pearson Education Ltd.

Pearson

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)

Pearson

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.
- These dimensions determine, to a large extent, the appropriate agent design and the applicability of each of the principal families of techniques for agent implementation

Properties of task environments

- Fully observable vs. partially observable:
- If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable. A task environment is effectively fully observable if the sensors detect all aspects that are relevant to the choice of action; 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.

Properties of task environments

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

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.
- One final note: 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").

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, 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 environments

- Static vs. dynamic:
- If the environment can change while an agent is deliberating (- To think carefully and often slowly, as about a choice to be made. -To consult with another or others in a process of reaching a decision), 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.

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.
- 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.

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.

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.
- Figure 2.6 lists the properties of a number of familiar environments.

Figure 2.6 Examples of task environments and their characteristics.

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

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??				

Pearson

© 2022 Pearson Education Ltd.

Environment types

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

Pearson

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??				

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??				

Chapter 2 © 2022 Pearson Education Ltd.

Pearson

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??	Yes	Yes	Yes	No

Pearson

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 Yes
Single-agent??		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

Chapter 2 © 2022 Pearson Education Ltd.

Pearson

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

Agent programs

- The agent programs that we design in our book all have the same skeleton: they take the current percept as input from the sensors and return an action to the actuators.
- For example, Figure 2.7 shows a rather trivial agent program that keeps track of the percept sequence and then uses it to index into a table of actions to decide what to do.

```
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.

Four basic kinds of agent programs

Four basic types in order of increasing generality:

- simple reflex agents
- reflex agents with state
- goal-based agents
- utility-based agents

All these can be turned into learning agents that can improve the performance of their components so as to generate better actions.

 © 2022 Pearson Education Ltd.

Pearson

Simple reflex agents

- The simplest kind of agent is the simple reflex agent. These agents select actions on the basis of the current percept, ignoring the rest of the percept history. For example, the vacuum agent is a simple reflex agent, because its decision is based only on the current location and on whether that location contains dirt.
- An agent program for this agent is shown in Figure 2.8.

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.

A simple reflex agent with rules

- A more 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. Figure 2.9 gives the 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.
- A condition–action rule: **if** car-in-front-is-braking **then** initiate-braking.

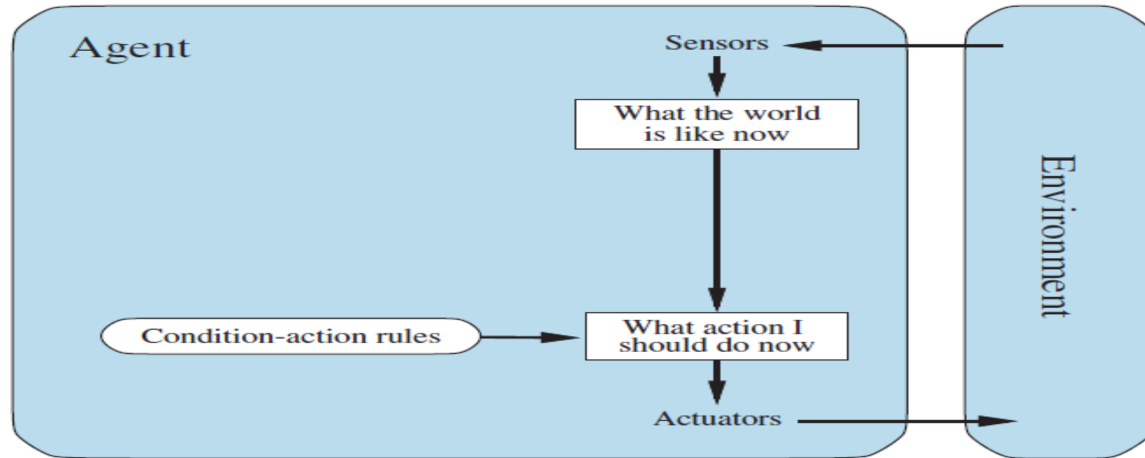


Figure 2.9 Schematic diagram of a simple reflex agent. We use rectangles to denote the current internal state of the agent's decision process, and ovals to represent the background information used in the process.

An agent program for acting according to rules

- An agent program for Figure 2.9 is shown in Figure 2.10. 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.
- 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.

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.

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*

```
(setq joe (make-agent :name 'joe :body (make-agent-body)
                      :program (make-reflex-vacuum-agent-program))

(defun make-reflex-vacuum-agent-program ()
  #'(lambda (percept)
      (let ((location (first percept)) (status (second percept)))
        (cond ((eq status 'dirty) 'Suck)
              ((eq location 'A) 'Right)
              ((eq location 'B) 'Left)))))
```


Pearson

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.
- we need some information about how the world changes over time. This knowledge about "how the world works" is called a transition model of the world.
- We need some information about how the state of the world is reflected in the agent's percepts. This kind of knowledge is called a sensor model.
- Together, the transition model and sensor model allow an agent to keep track of the state of the world—to the extent possible given the limitations of the agent's sensors. An agent that uses such models is called a model-based agent.

A model-based reflex agent with state

- Figure 2.11 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.

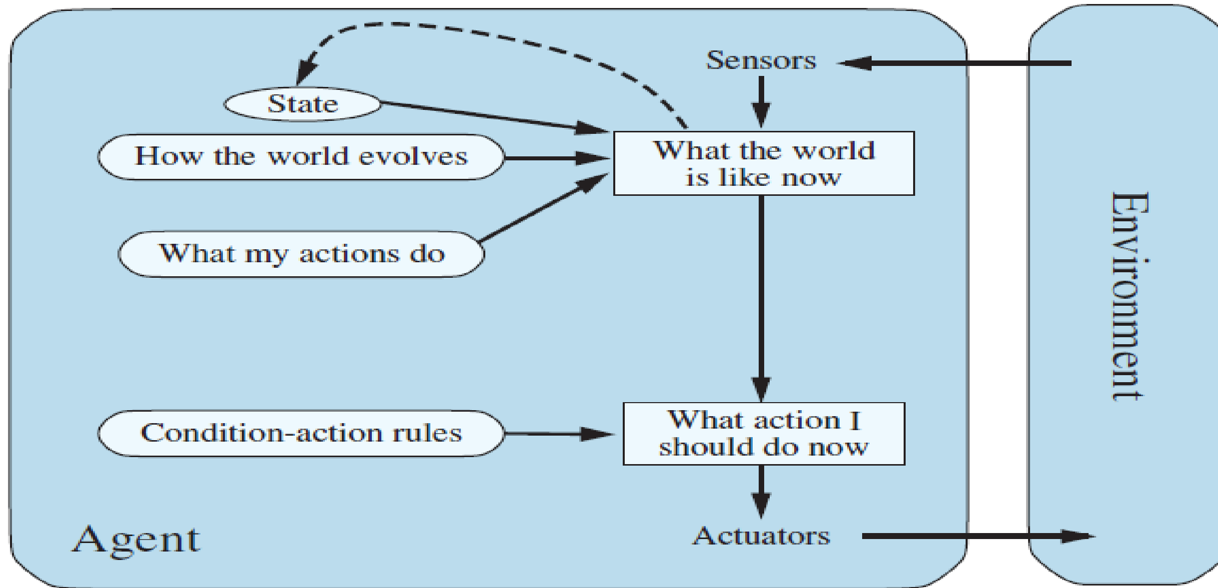


Figure 2.11 A model-based reflex agent.

A model-based reflex agent

- The agent program is shown in Figure 2.12. The interesting part is the function UPDATE-STATE, which is responsible for creating the new internal state description. The details of how models and states are represented vary widely depending on the type of environment and the particular technology used in the agent design.

```
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.

Example

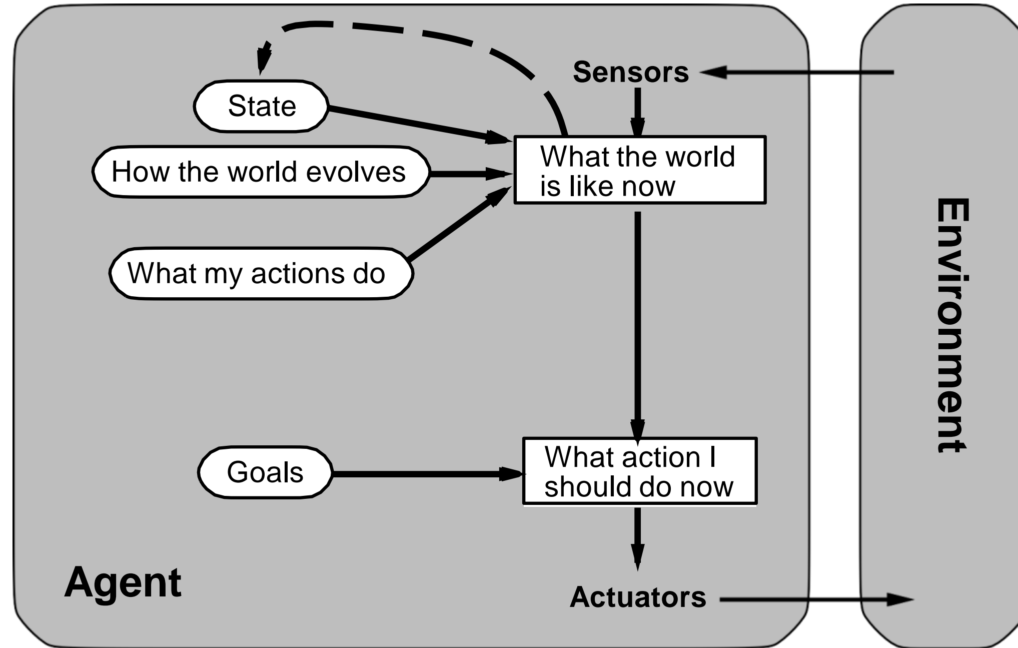
function Reflex-Vacuum-Agent([*location*,*status*]) returns an action
static: *last_A*, *last_B*, numbers, initially ∞
 if *status* = *Dirty* then . . .

```
(defun make-reflex-vacuum-agent-with-state-program () (let ((last-A
       infinity)          (last-B infinity)) #'(lambda     (percept)
     (let ((location (firstpercept)) (status (second percept))) (incf   last-A)
        (incf     last-B)
      (cond
       ((eq status 'dirty)
        (if (eq location 'A) (setq last-A 0) (setq last-B 0)) 'Suck)

((eq   location   'A)  (if  (>  last-B  3)  'Right   'NoOp))
((eq   location   'B)  (if  (>  last-A  3)  'Left  'NoOp)))))))
```


Pearson

Model-based agents



© 2022
Pearson
Education
Ltd.

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.
- Figure 2.13 shows the goal-based agent's structure.

A model-based, goal-based agent.

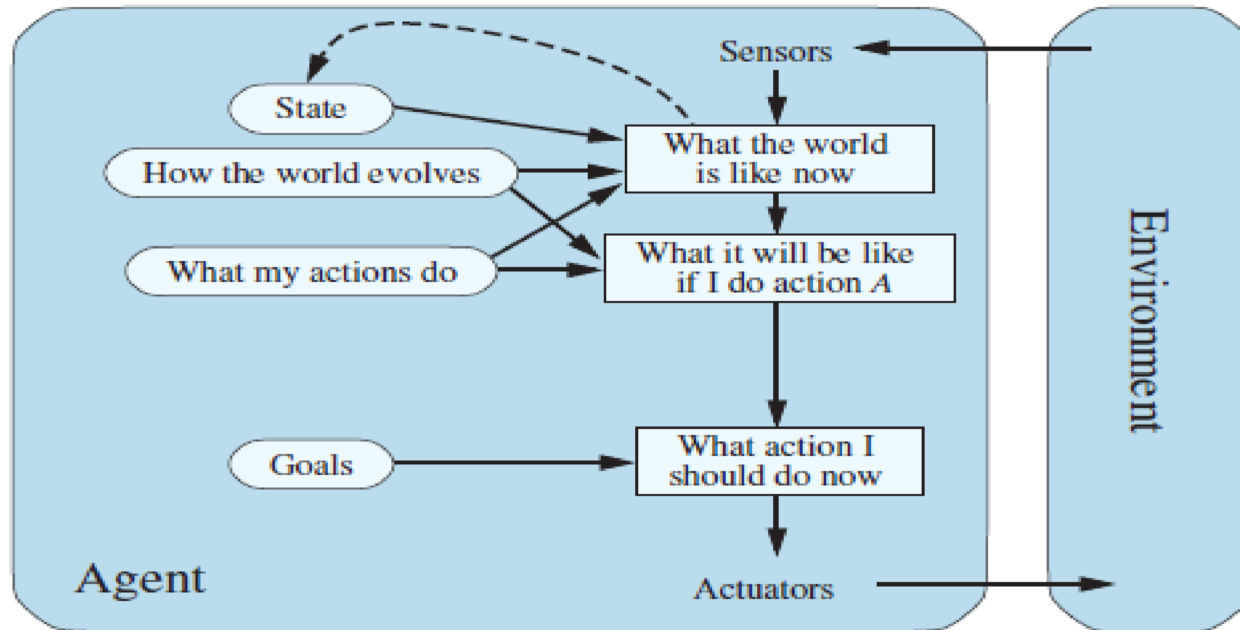


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

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.
- We have already seen that a performance measure assigns a score to any given sequence of environment states.
- 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.
- The utility-based agent structure appears in Figure 2.14.

A model-based, utility-based agent.

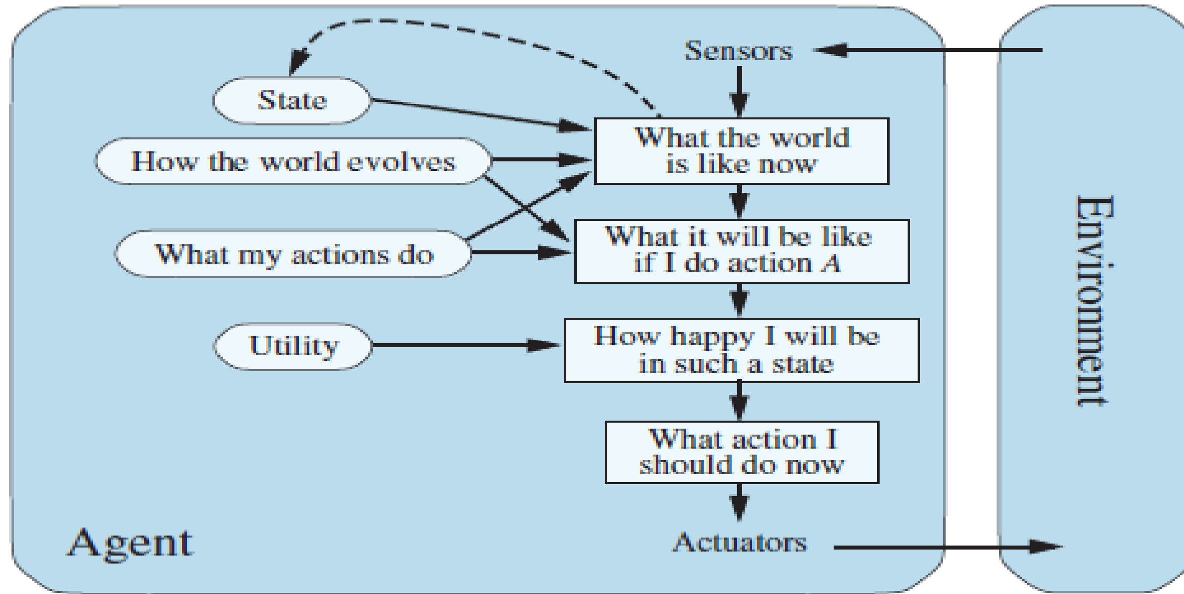


Figure 2.14 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome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

Learning agents

- Turing (1950) considers the idea of actually programming his intelligent machines by hand. He estimates how much work this might take and concludes, “Some more expeditious method seems desirable.” The method he proposes is to build learning machines and then to teach them. In many areas of AI, this is now the preferred method for creating state-of-the-art systems.
- Any type of agent (model-based, goal-based, utility-based, etc.) can be built as a learning agent (or not).
- Learning has another advantage, as we noted earlier: it allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow.

Learning agents

- A learning agent can be divided into four conceptual components, as shown in Figure 2.15.
- 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.

Learning agents

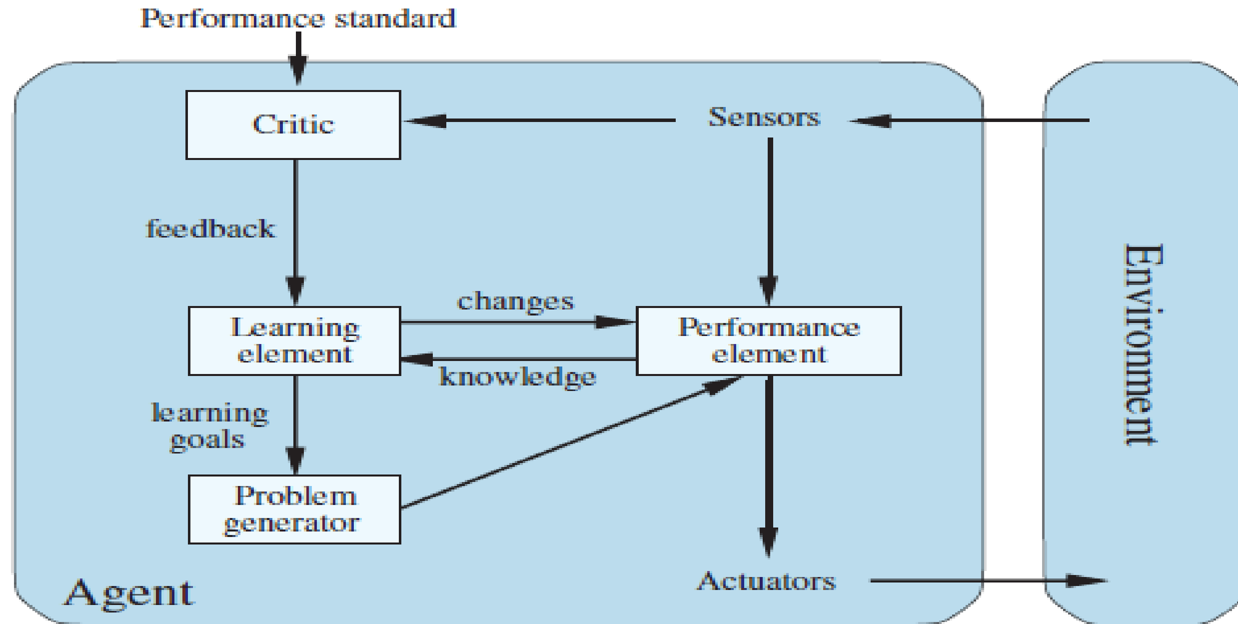
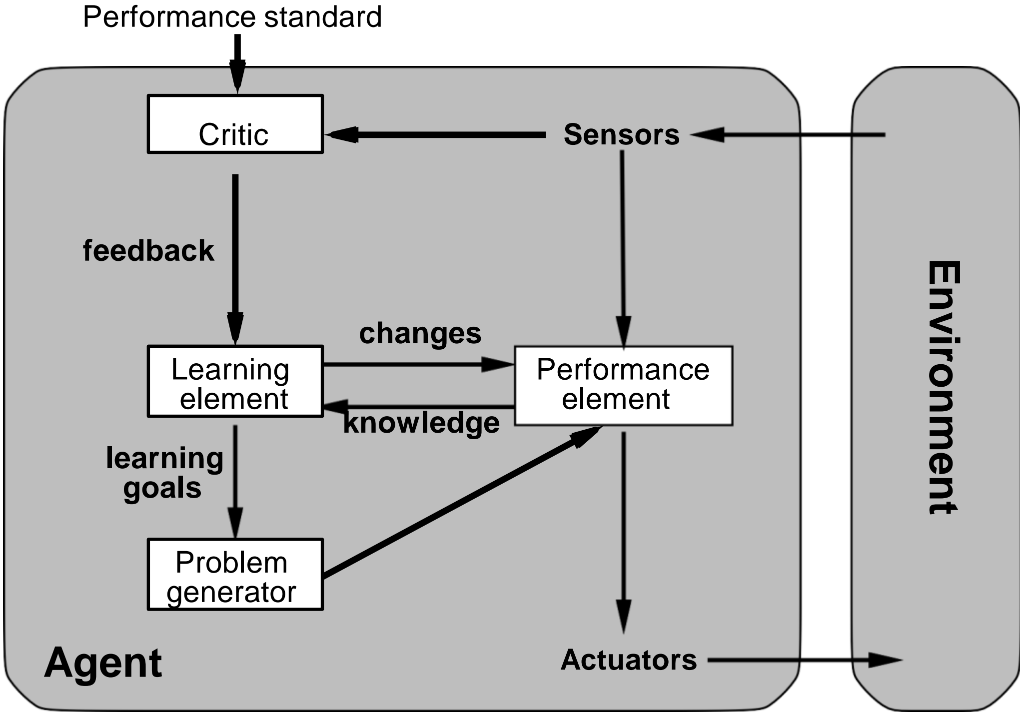


Figure 2.15 A general learning agent. The "performance element" box represents what we have previously considered to be the whole agent program. Now, the "learning element" box gets to modify that program to improve its performance.

Learning agents



Pearson

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

Pearson