INTELLIGENT AGENTS

Outline

- Agents and Environments
- Good Behavior and Rationality
- Nature of Environments
- Structure of Agents

Based on the textbook by Stuart Russell & Peter Norvig:

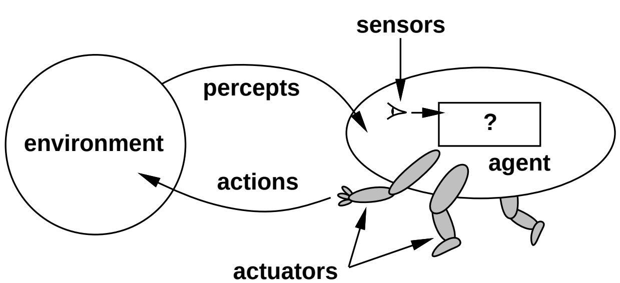
Artificial Intelligence, A Modern Approach. (2nd Edition)

Chapter 2

1. AGENTS AND ENVIRONMENTS

Definition. Russell and Norvig define agents as follows:

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



Examples

A physical robot

- Sensors: video camera, laser scanner, microphone, . . .
- Actuators: motor, switch, display, speaker, . . .

A software robot (softbot)

- Percepts: encoded bit strings
- Sensors and actuators:
  calls to operating system, libraries or other programs
- Calls to sensor programs provide input for the agent.

Mapping Percept Sequences to Actions

- Agent's behavior depends only on its percept sequence to date.
- An agent can be designed by

  > *"specifying which action an agent ought to take in response to any given percept sequence".*

- Such a mapping (**agent function**) can be represented as a table or as an **agent program**.

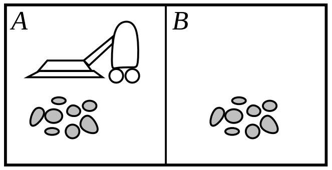
Example. Consider a calculator agent that computes square-roots of positive integers (accurate to 15 decimals).

Approach 1: store square roots in a very large table.

Approach 2: implement the ideal mapping as a program.

The latter approach is clearly more compact and flexible.

Example. 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*
...	...

2. GOOD BEHAVIOR AND RATIONALITY

- A *rational agent* should do the right thing, but *how* and *when* do we evaluate agent's success?
- A *performance measure* determines how successful an agent is (by an outside observer).

Problems:

- Self-deception: humans typically say they did not really want something after they are unsuccessful at getting it.
- Malpractice if performance is measured only instantly.
- You get what you ask for!
- The measure should depend on effects on the environment.

Rational Agents

Rationality depends on four things:

- The performance measure which 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 (complete perceptual history).

Definition. (*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.

Omniscience vs. Rationality

An omniscient agent

- knows the *actual* outcomes of its actions and acts accordingly; and
- is impossible in reality.

Example. A person is crossing a street, as (s)he noticed a friend across the street and there is no traffic nearby.

Is this person acting rationally if (s)he is crashed by a cargo door falling off a passing airplane?

☞ Rationality: expected success given what has been perceived.

Information Gathering

Example. Often begin rational requires performing actions in order to acquire information about the environment.

➤ For instance, crossing a street without looking is too risky.

Example. A clock can be thought as a simple (even degenerate) agent that keeps moving its hands (or displaying digits) in the proper way.

- This can be thought as rational action given what kind of functionality one expects from a clock in general.
- However, many clocks are unable to take changing time zones into account automatically. This is quite acceptable if the clock does not have a mechanism for perceiving time zones.

Autonomy

➤ An agent lacks autonomy if its actions depend solely on its built-in knowledge about the environment.

➤ A system is *autonomous* to the extent that its behavior is determined by its own experience.

➤ Flexible operation in a variety of environments demands ability to learn (in addition to initial knowledge).

Example. After digging its nest and laying its eggs, a dung beetle fetches a ball of dung to plug the entrance.

➤ Even if the ball is removed from its grasp en route, the beetle continues and mimics the procedure to the very end.

3. NATURE OF ENVIRONMENTS

➤ A **task environment** specifies a "problem" to which a rational agent is a "solution".

➤ Task environments can be roughly specified by giving a **PEAS** (Performance, Environment, Actuators, Sensors) description.

Example. Task environment for an automated taxi.

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

Examples of PEAS Descriptions

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient minimize costs, lawsuits	Patient, hospital staff	Display questions, tests, diagnoses, treatments	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display categorization of scene	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	Maximize purity, yield, safety	Refinery, operators	Valves, pumps heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Maximize student's score on test	Set of students, testing agency	Display exercises, suggestions, corrections	Keyboard entry

Properties of Task Environments

Environments can be categorized by several aspects such as

- *Fully vs. partially observable* state of the environment
  Also: *effectively fully observable*
- *Deterministic vs. stochastic* outcomes of agent's actions
- *Episodic vs. sequential*
- *Static vs. dynamic*
  Also: *semidynamic* (performance degrades over time)
- *Discrete vs. continuous*
- *Single agent vs. multiagent* (competitive or cooperative)

Examples. Analyzing properties of a number of familiar environments.

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

- Some of the properties are dependent on how the environments and agents are conceptualized.

4. STRUCTURE OF AGENTS

- The goal is to design an *agent program* which implements the mapping from percepts to actions.
- An *architecture* is a computing device that makes percepts available, runs the program and feeds action choices to actuators.
- Summarizing: **agent** = **architecture** + **program**.

Agent Programs

A skeleton for agent programs:

- A single percept is obtained as input.
- Memory is used for storing the percept history (if necessary).
- The program chooses and outputs an action to be executed next.

```
function SKELETON-AGENT(percept) returns action
   static: memory, the agent's memory of the world

   memory ← UPDATE-MEMORY(memory, percept)
   action ← CHOOSE-BEST-ACTION(memory)
   memory ← UPDATE-MEMORY(memory, action)
   return action
```

- The performance measure is not a part of the program.

Using Lookup Tables

- An agent program that looks up the action from a table:

```
function TABLE-DRIVEN-AGENT(percept) returns action
   static: percepts, a sequence, initially empty
           table, a table, indexed by percept sequences, initially fully specified

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

- Drawbacks of lookup table agents:
  1. The lookup table becomes easily very large (a chess playing agent would need a table with 35^{100} entries).
  2. The table is difficult to build and maintain.
  3. The resulting agent does not have autonomy at all.
  4. It would take forever to learn the right values for all entries.

Different Kinds of Agent Programs

In the sequel, we will consider four kinds of agent programs:

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

Then we discuss a general way to incorporate *learning* into these.

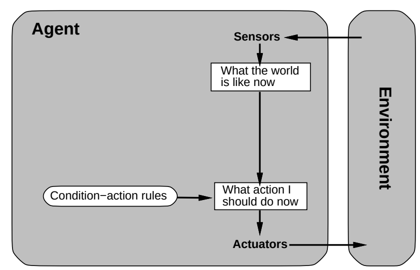
Simple Reflex Agents

- *Condition-action rules* provide a way to represent common regularities appearing in input/output associations:

  if car-in-front-is-braking **then** initialize-braking

- It is even possible to learn such rules on the fly.
- Humans also have many such connections some of which are *learned responses* and some of which are *innate reflexes* (such as eye blinking in order to protect the eye).
- Mimicking reflexes of living creatures, a **reflex agent** chooses the next action on the basis of the current percept.
- No track of the world/environment is kept.

- The structure of a simple reflex agent as a schematic diagram and the corresponding skeletal agent program:




```
function SIMPLE-REFLEX-AGENT(percept) returns action
   static: rules, a set of condition-action rules

   state ← INTERPRET-INPUT(percept)
   rule ← RULE-MATCH(state, rules)
   action ← RULE-ACTION[rule]
   return action
```

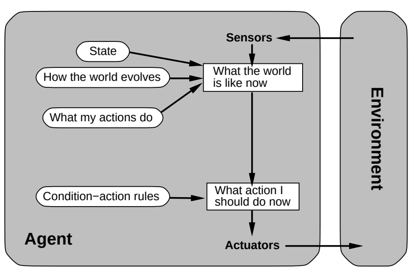
- Rules provide an efficient representation, but one problem is that decision making is seldom possible on the basis of a single percept.

Model-based Reflex Agents

- The choice of actions may depend on the entire percept history.
- Sensors do not necessarily provide access to the complete state of the environment.
- The agent keeps track of the world by extracting relevant information from percepts and storing it in its memory.
- Using a **model** of the environment, the agent may try to estimate
  1. how the environment evolves in the (near) future, and
  2. how the environment is affected by the agent's actions.

Example. In our taxi driving example, actions may depend on the state, e.g. the position of an overtaking car.

- A schematic diagram and a skeletal agent program for a model-based reflex agent:



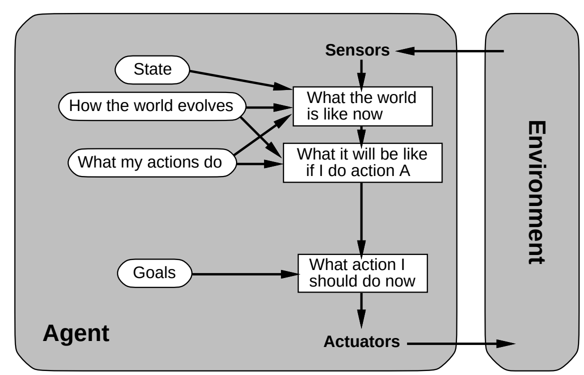

```
function REFLEX-AGENT-WITH-STATE(percept) returns action
   static: state, a description of the current world state
           rules, a set of condition-action rules

   state ← UPDATE-STATE(state, percept)
   rule ← RULE-MATCH(state, rules)
   action ← RULE-ACTION[rule]
   state ← UPDATE-STATE(state, action)
   return action
```


Goal-based agents

- Knowing about the current state of the environment is not necessarily enough for deciding what to do.
- In addition, the agent may need **goals** to distinguish which situations are desirable and which are not.
- Goal information can be combined with the agent's knowledge about the results of possible actions in order to choose an action leading to a goal.
- Problem: goals are not necessarily achievable by a single action.
- **Search** and **planning** are subfields of AI devoted to finding actions sequences that achieve the agent's goals.

- A schematic diagram for a goal-based agent:

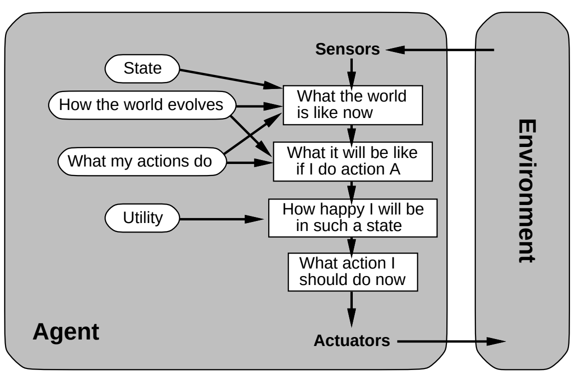


- Additional flexibility compared to previous designs: the behavior of a goal-based agent can be changed by changing its goal(s).

Utility-based agents

- Goals alone are not sufficient for decision making if there are several ways of achieving them.
- Further problem: agents may have several conflicting goals that cannot be achieved simultaneously.
- If an agent prefers one world state to another state then the former state has higher utility for the agent.
- Utility is a function that maps a state onto a real number.
- A utility function can be used for (i) choosing the best plan, (ii) resolving conflicts among goals, and (iii) estimating the successfulness of an agent if the outcomes of actions are uncertain.

- A schematic diagram for a utility-based agent:

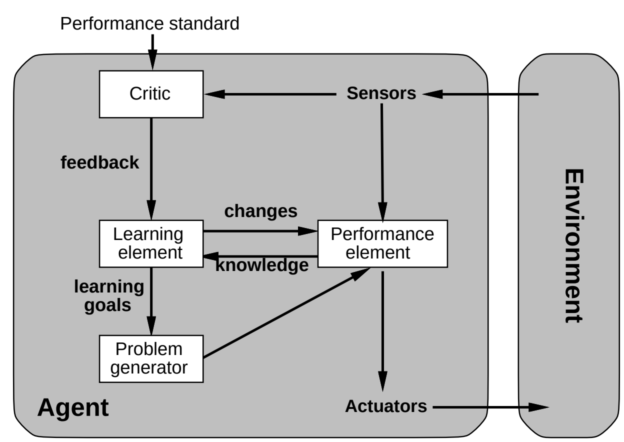


- An agent that possesses an *explicit* utility function can make rational decisions, but may have to compare the utilities achieved by different courses of actions.

Learning Agents

- A **learning element** is responsible for improving the **performance element**, which corresponds to an entire agent.
- The learning element gets feedback from a **critic** on the performance of the agent.
- A **problem generator** suggests actions that will lead to new and informative experiences.
- Sometimes the percepts of the agent include *rewards* or *penalties* (such as pain and hunger) that can be utilized in learning.

- A schematic diagram for a learning agent:



Programs Simulating Environments

```
procedure RUN-ENVIRONMENT(state, UPDATE-FN, agents, termination)
  inputs: state, the initial state of the environment
          UPDATE-FN, function to modify the environment
          agents, a set of agents
          termination, a predicate to test when we are done

  repeat
     for each agent in agents do
        PERCEPT[agent] ← GET-PERCEPT(agent, state)
     end
     for each agent in agents do
        ACTION[agent] ← PROGRAM[agent](PERCEPT[agent])
     end
     state ← UPDATE-FN(actions, agents, state)
  until termination(state)
```

- Agents are typically designed to work correctly in a class of environments (that has to be covered by a simulator somehow).
- Agent programs should not have other access than percepts to the state of the program simulating their environment!

- The performance of agents can be measured by inserting special measurement code to simulator programs.

```
function RUN-EVAL-ENVIRONMENT(state, UPDATE-FN, agents,
                             termination, PERFORMANCE-FN) returns scores
  local variables: scores, a vector the same size as agents, all 0

  repeat
     for each agent in agents do
        PERCEPT[agent] ← GET-PERCEPT(agent, state)
     end
     for each agent in agents do
        ACTION[agent] ← PROGRAM[agent](PERCEPT[agent])
     end
     state ← UPDATE-FN(actions, agents, state)
     scores ← PERFORMANCE-FN(scores, agents, state)
  until termination(state)
  return scores                                          /* change */
```


SUMMARY

- An agent program maps a percept (sequence) to an action.
- Agent = architecture + agent program.
- A rational agent tries to maximize its performance measure.
- Task environments can be described by PEAS descriptions.
- Various agent types: reflex agents with(out) internal state, goal-based agents, utility-based agents.
- Important aspects of agent program design: efficiency, compactness, flexibility.

QUESTIONS

- Analyze soccer playing agents by writing down
  1. a PEAS description and
  2. the main properties of the environment.
- Consider the following designs for soccer playing agents:
  1. Simple reflex agent
  2. Model-based reflex agent
  3. Agent with explicit goals
  4. Utility-based agent
  5. Learning agent

  What kind of functionality can be implemented in terms of these?