Agents and Environments

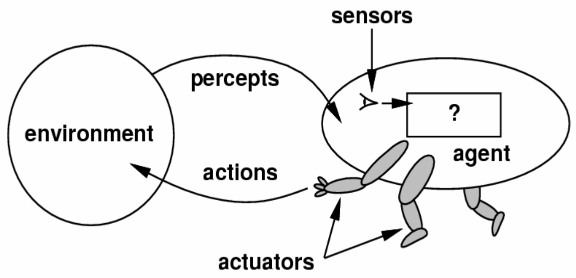
Berlin Chen
Department of Computer Science & Information Engineering
National Taiwan Normal University

Reference:

1. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Chapter 2 & Teaching Material

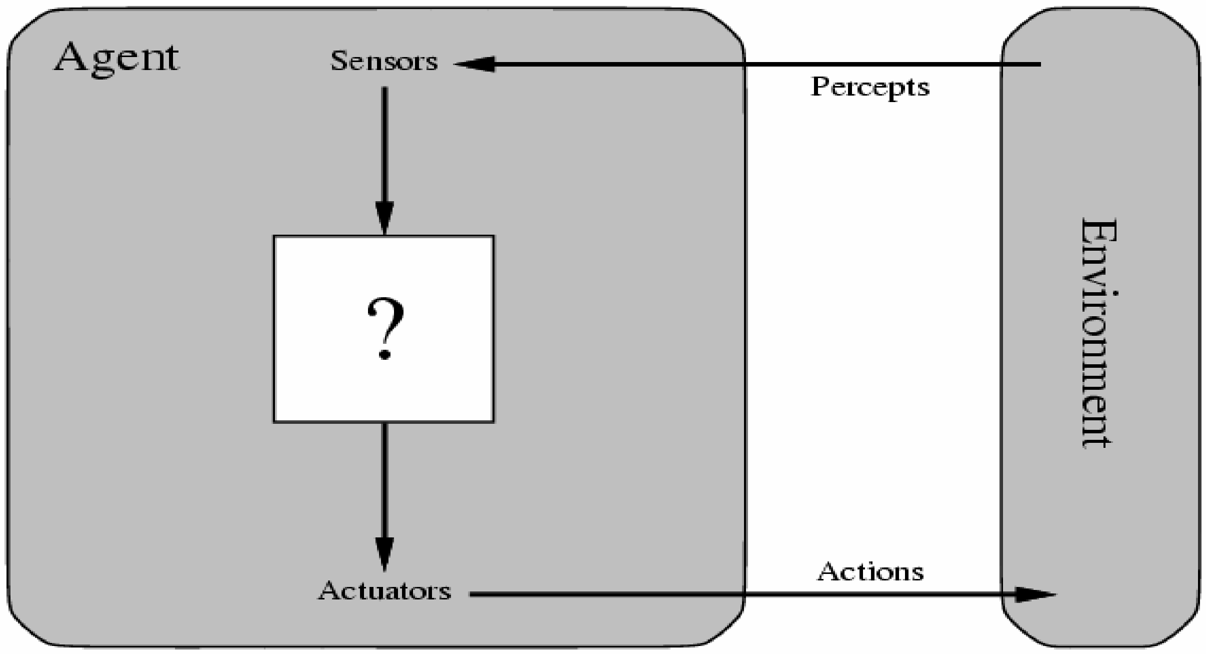
What is an Agent

- An agent interacts with its environments
  - Perceive through sensors
    - Human agent: eyes, ears, nose etc.
    - Robotic agent: cameras, infrared range finder etc.
    - Soft agent: receiving keystrokes, network packages etc.
  - Act through actuators
    - Human agent: hands, legs, mouse etc.
    - Robotic agent: arms, wheels, motors etc.
    - Soft agent: display, sending network packages etc.



- A rational agent is
  - One that does the right thing
  - Or one that acts so as to achieve best expected outcome

Agent and Environments



Assumption: every agent can perceive its own actions

Agent and Environments (cont.)

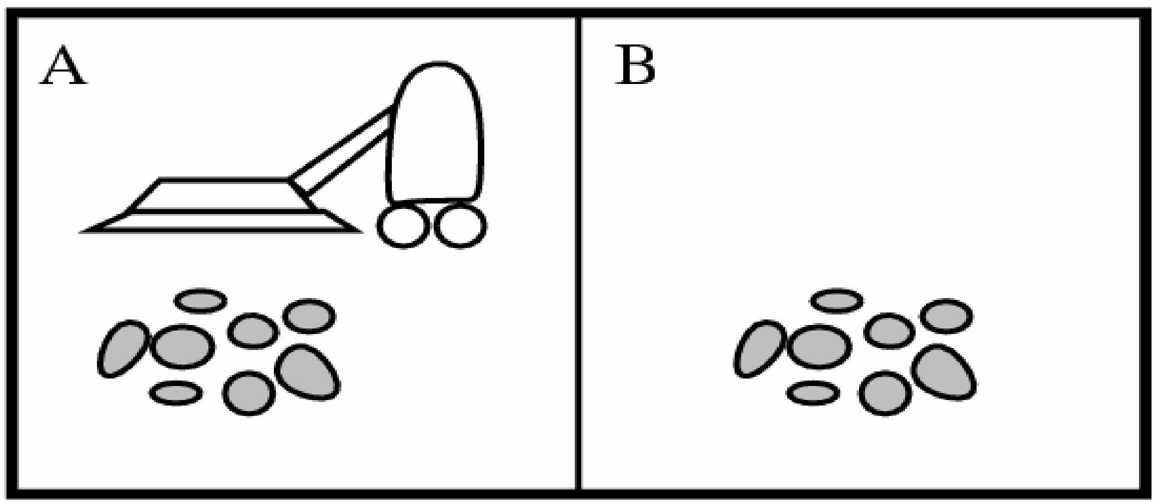
- Percept (*P*)
  - The agent's perceptual inputs at any given time
- Percept sequence (*P**)
  - The complete history of everything the agent has ever perceived
- Agent function
  - A mapping of any given percept sequence to an action

$$f : P^*\left(P_0, P_1, \ldots, P_n\right) \rightarrow A$$

  - Agent function is implemented by an agent program
- Agent program
  - Run on the physical agent architecture to produce f

Example: Vacuum-Cleaner World

- A made-up world
- Agent (vacuum cleaner)
  - Percepts:
    - Square locations and contents, e.g. [A, Dirty], [B, Clean]
  - Actions:
    - Right, Left, Suck or NoOp



A Vacuum-Cleaner Agent

- Tabulation of agent functions

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 simple 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*

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 (to be most successful)**, given the evidence provided by the percept sequence to date and whatever built-in knowledge the agent has

  – Performance measure
  – Percept sequence
  – Prior knowledge about the environment
  – Actions

Performance Measure for Rationality

- Performance measure
  - Embody the criterion for success of an agent's behavior
- Subjective or objective approaches
  - Objective measure is preferred
  - E.g., in the vacuum-cleaner world:

    amount of dirt cleaned up

    or the electricity consumed per time step

    or average cleanliness over time

    (which is better?)
- How and when to evaluate?
- Rationality vs. perfection (or omniscience)
  - Rationality => exploration, learning and autonomy

A rational agent should be autonomous!

Task Environments

- When thinking about building a rational agent, we must specify the task environments
- The **PEAS** description
  - Performance
  - Environment
  - Actuators
  - Sensors

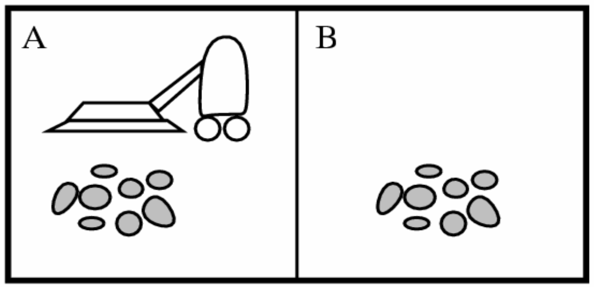
Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits correct destination	Roads, other traffic, pedestrians, customers places, countries	Steering, accelerator, brake, signal, horn, display talking with passengers	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Figure 2.4 PEAS description of the task environment for an automated taxi.

Task Environments (cont.)

- Properties of task environments: Informally identified (categorized) in some dimensions

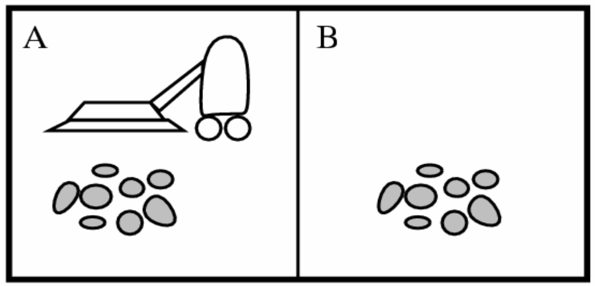
  – Fully observable vs. partially observable
  – Deterministic vs. stochastic
  – Episodic vs. sequential
  – Static vs. dynamic
  – Discrete vs. continuous
  – Single agent vs. multiagent



Fully Observable vs. Partially Observable

- Fully observable
  - Agent can access to the complete state of the environment at each point in time
  - Agent can detect all aspect that are relevant to the choice of action

- E.g. (Partially observable)
  - A vacuum agent with only local dirt sensor doesn’t know the situation at the other square
  - An automated taxi driver can’t see what other drivers are thinking



Deterministic vs. Stochastic

- Deterministic
  - The next state of the environment is completely determined by the current state and the agent's current action
- E.g.
  - The vacuum world is deterministic, but stochastic when randomly appearing dirt (due to unreliable suction mechanism)
  - The taxi-driving environment is stochastic: never predict the behavior of traffic exactly
- Strategic
  - Nondeterministic because of the other agents' action

Episodic vs. Sequential

- Episodic
  - The agent’s experience is divided into atomic episode
    - Each episode consists of the agent perceiving and then performing a single action
  - The next episode doesn’t depend on the actions taken in previous episode (depend only on episode itself)
- E.g.
  - Classification task: Spotting defective parts on assembly line is episodic
  - Chess-playing and taxi-driving case are sequential

Static vs. Dynamic

- Dynamic
  - The environment can change while the agent is deliberating (仔細考慮)
  - Agent is continuously asked what to do next
    - Thinking means do “nothing”

- E.g.
  - Taxi-driving is dynamic
    - Other cars and itself keep moving while the agent dithers about (躊躇) what to do next
  - Crossword puzzle is static

- Semi-dynamic
  - The environment doesn’t change but the agent’s performance score does (time passage degrades the agent’s performance)
  - E.g., chess-playing with a clock

Discrete vs. Continuous

- The environment states (continuous-state ?) and the agent's percepts and actions (continuous-time?) can be either discrete and continuous

- E.g.
  - Taxi-driving is a continuous-state (location, speed, etc.) and continuous-time (steering, accelerating, camera, etc.) problem

Single agent vs. Multi-agent

- Single-agent
  - E.g., crossword puzzle, Sudoku (數獨), etc.
- Multi-agent
  - Multiple agents existing in the environment
  - How a entry may be viewed as an agent ?

- Two kinds of multi-agent environment
  - Cooperative
    - E.g., taxing-driving is partially cooperative (avoiding collisions, etc.)
    - Communication may be required
  - Competitive
    - E.g., chess-playing
    - Stochastic behavior is rational

Task Environments (cont.)

- Examples

Task Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Crossword puzzle	Fully	Deterministic	Sequential	Static	Discrete	Single
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Poker	Partially	Strategic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partially	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Image-analysis	Fully	Deterministic	Episodic	Semi	Continuous	Single
Part-picking robot	Partially	Stochastic	Episodic	Dynamic	Continuous	Single
Refinery controller	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Interactive English tutor	Partially	Stochastic	Sequential	Dynamic	Discrete	Multi

Figure 2.6 Examples of task environments and their characteristics.

- The most hardest case
  - Partially observable, stochastic, sequential, dynamic, continuous, multi-agent

The Structure of Agents

- How do **the insides** of agents work
  - In addition their behaviors

- A general agent structure

  Agent = Architecture + Program

- Agent program
  - Implement the agent function to map percepts (inputs) from the sensors to actions (outputs) of the actuators
    - Need some kind of approximation ?
  - Run on a specific architecture

- Agent architecture
  - The computing device with physical sensors and actuators
  - E.g., an ordinary PC or a specialized computing device with sensors (camera, microphone, etc.) and actuators (display, speaker, wheels, legs etc.)

The Structure of Agents (cont.)

- Example: the table-driven-agent program

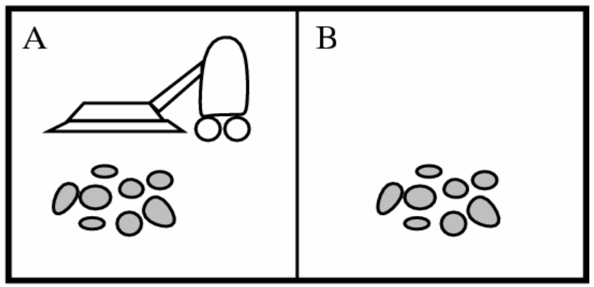
```
function TABLE-DRIVEN-AGENT(percept) returns an action
  static: percepts, a sequence, initially empty
          table, a table of actions, indexed by percept sequences, initially fully specified

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

  - Take the current percept as the input
  - The "table" explicitly represent the agent functions that the agent program embodies
  - Agent functions depend on the entire percept sequence

The Structure of Agents (cont.)

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, $Dirty$]	$Suck$
⋮	⋮



The Structure of Agents (cont.)

- Steps done under the agent architecture

    1. Sensor's data $\rightarrow$ Program inputs (Percepts)
    2. Program execution
    3. Program output $\rightarrow$ Actuator's actions

- Kinds of agent program
    - *Table-driven agents -> doesn't work well!*
    - Simple reflex agents
    - Model-based reflex agents
    - Goal-based agents
    - Utility-based agents

Table-Driven Agents

- Agents select actions based on the entire percept sequence (as that shown previously in P. 19)
- Table lookup size: $\sum_{t=1}^{T} |P|^t$
  - *P*: possible percepts
  - *T*: life time

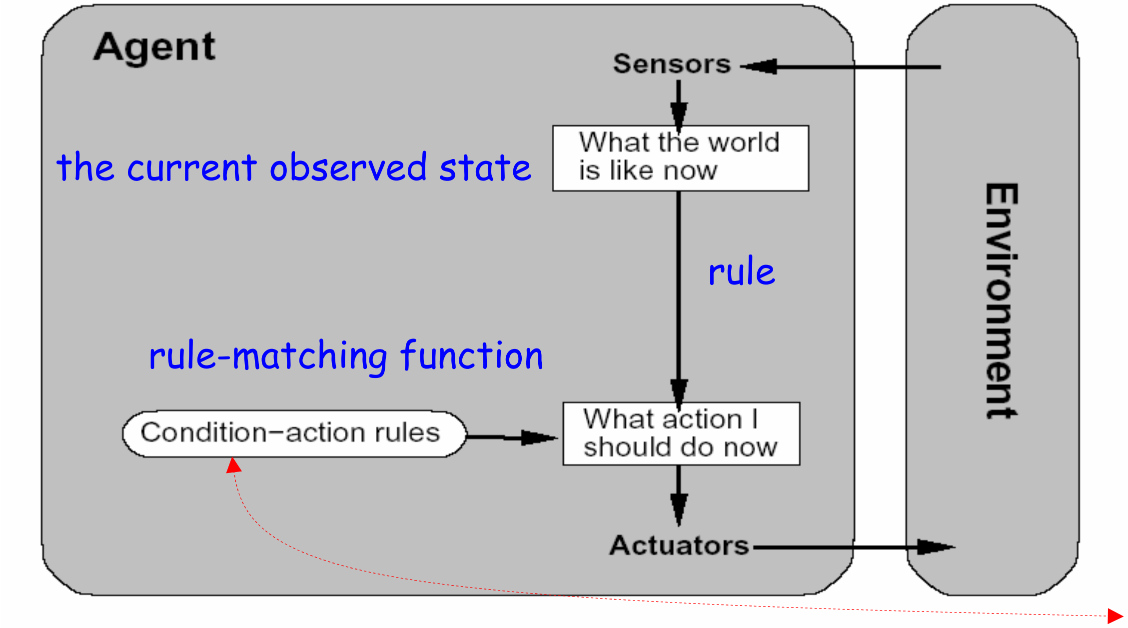
- Problems with table-driven agents
  - Memory/space requirement
  - Hard to learn from the experience
  - Time for constructing the table

How to write an excellent program to produce rational behavior from a small amount of code rather than from a large number of table entries

- Doomed to failure

Simple Reflex Agents

- Agents select actions based on the current percept, ignoring the rest percept history
  - Memoryless
  - Respond directly to percepts



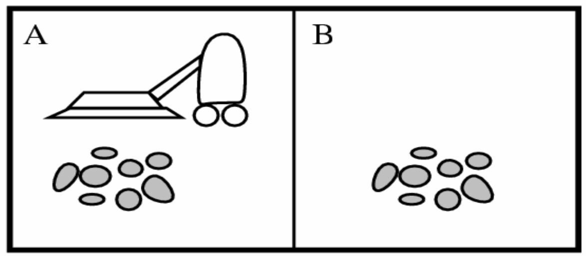
e.g., **If** car-in-front-is-braking **then** initiate-braking

  - Rectangles: internal states of agent's decision process
  - Ovals: background information used in decision process

Simple Reflex Agents

- Example: the vacuum agent introduced previously
  - It’s decision is based only on the current location and on whether that contains dirt
  - Only 4 percept possibilities/states (instead of 4^T)

[*A, Clean*]
[*A, Dirty*]
[*B, Clean*]
[*B, Dirty*]



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*

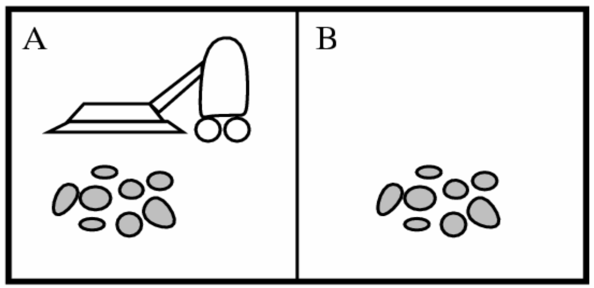
Simple Reflex Agents (cont.)

- Problems with simple reflex agents
  - Work properly if the environment is fully observable
  - Couldn't work properly in partially observable environments
  - Limited range of applications

- Randomized vs. deterministic simple reflex agent
  - E.g., the vacuum-cleaner is deprived of its location sensor
    - Randomize to escape infinite loops

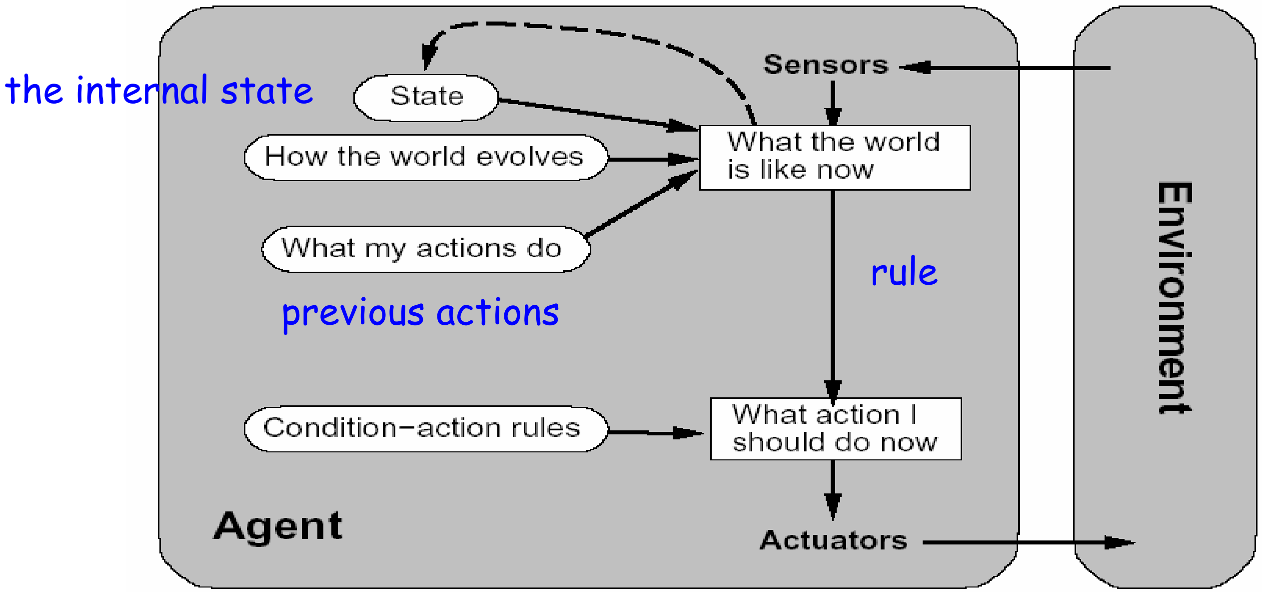
```
function SIMPLE-REFLEX-AGENT(percept) returns an action
   static: rules, a set of condition–action rules

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



Model-based Reflex Agents

- Agents maintain internal state to track aspects of the world that are not evident in the current state
  - Parts of the percept history kept to reflect some of the unobserved aspects of the current state
  - Updating internal state information require knowledge about
    - Which perceptual information is significant
    - How the world evolves independently
    - How the agent's action affect the world



Model-based Reflex Agents (cont.)

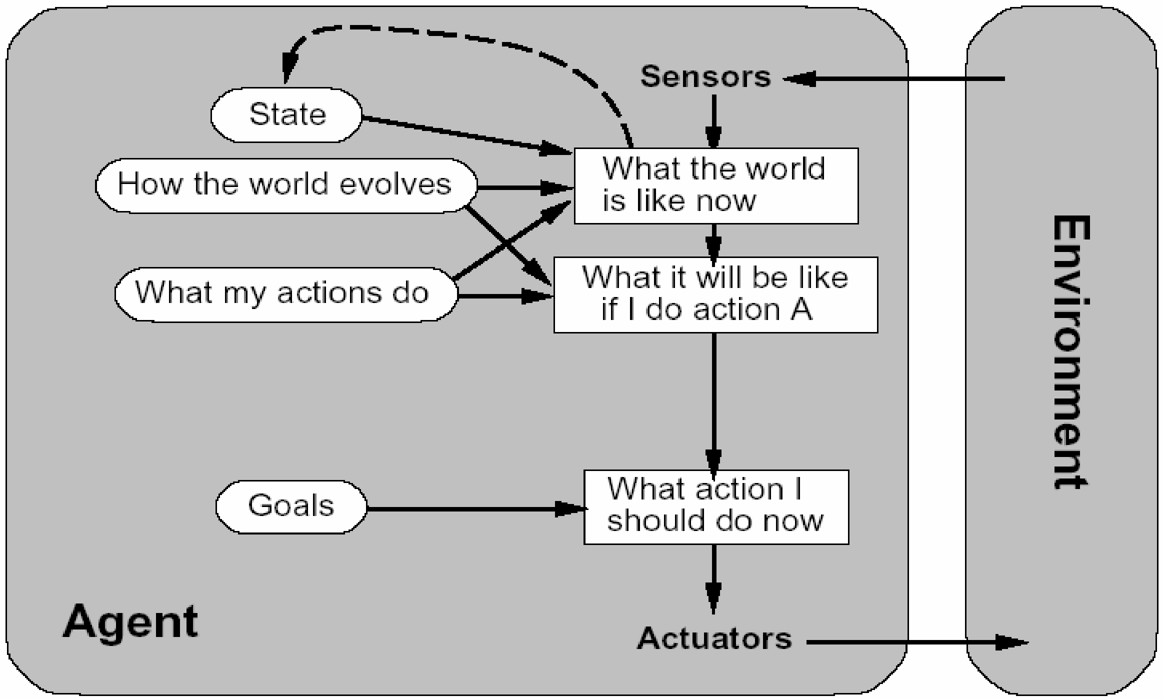
```
function REFLEX-AGENT-WITH-STATE(percept) returns an action
   static: state, a description of the current world state
           rules, a set of condition–action rules
           action, the most recent action, initially none

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

Goal-based Agents

- The action-decision process involves some sort of goal information describing situations that are desirable
  - Combine the goal information with the possible actions proposed by the internal state to choose actions to achieve the goal
  - **Search** and **planning** in AI are devoted to finding the right action sequences to achieve the goals

What will happen if I do so? Consideration of the future

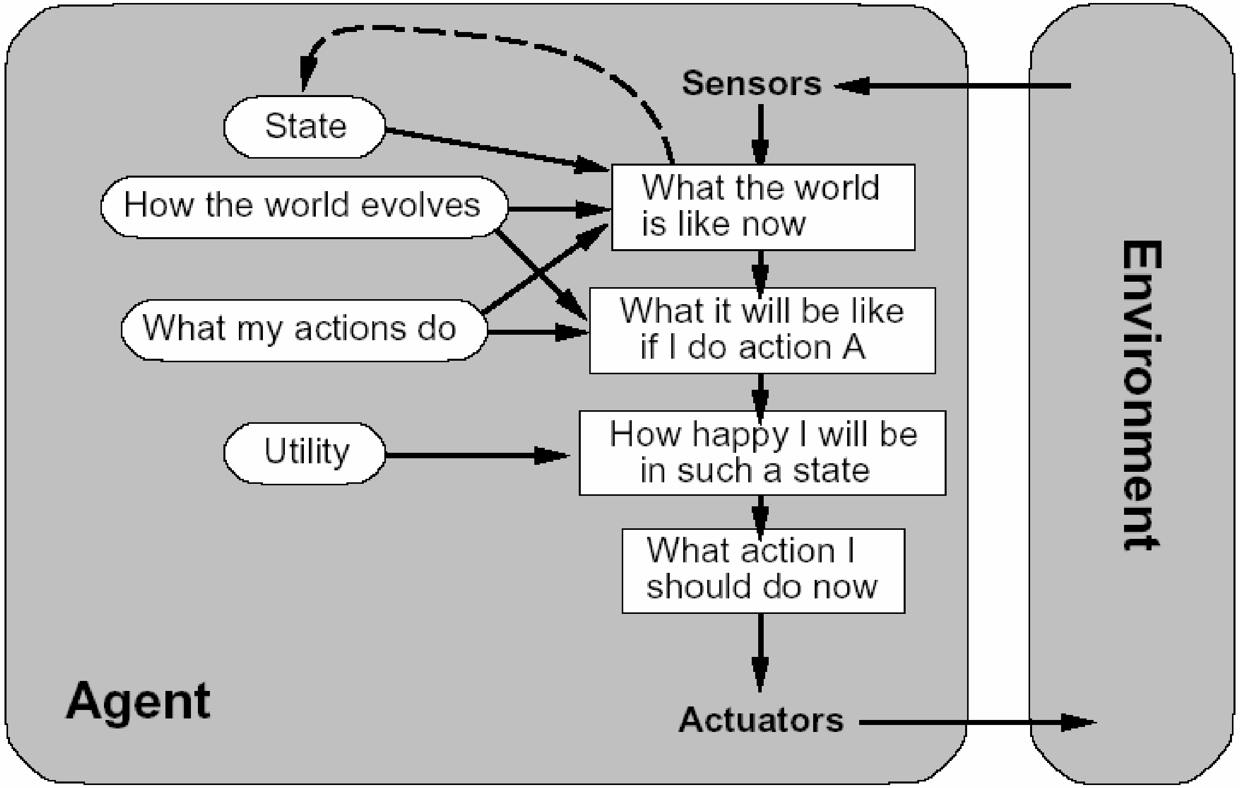


Utility-based Agents

- Goal provides a crude binary distinction between “happy” and “unhappy” sates
- Utility: maximize the agents expected happiness
  - E.g., quicker, safer, more reliable for the taxis-driver agent
- Utility function
  - Map a state (or a sequence of states) onto a real number to describe to degree of happiness
  - Explicit utility function provides the appropriate tradeoff or uncertainties to be reached of several goals
    - Conflict goals (speed/safety)
    - Likelihood of success among the goals

Make rational decisions

Utility-based Agents (cont.)

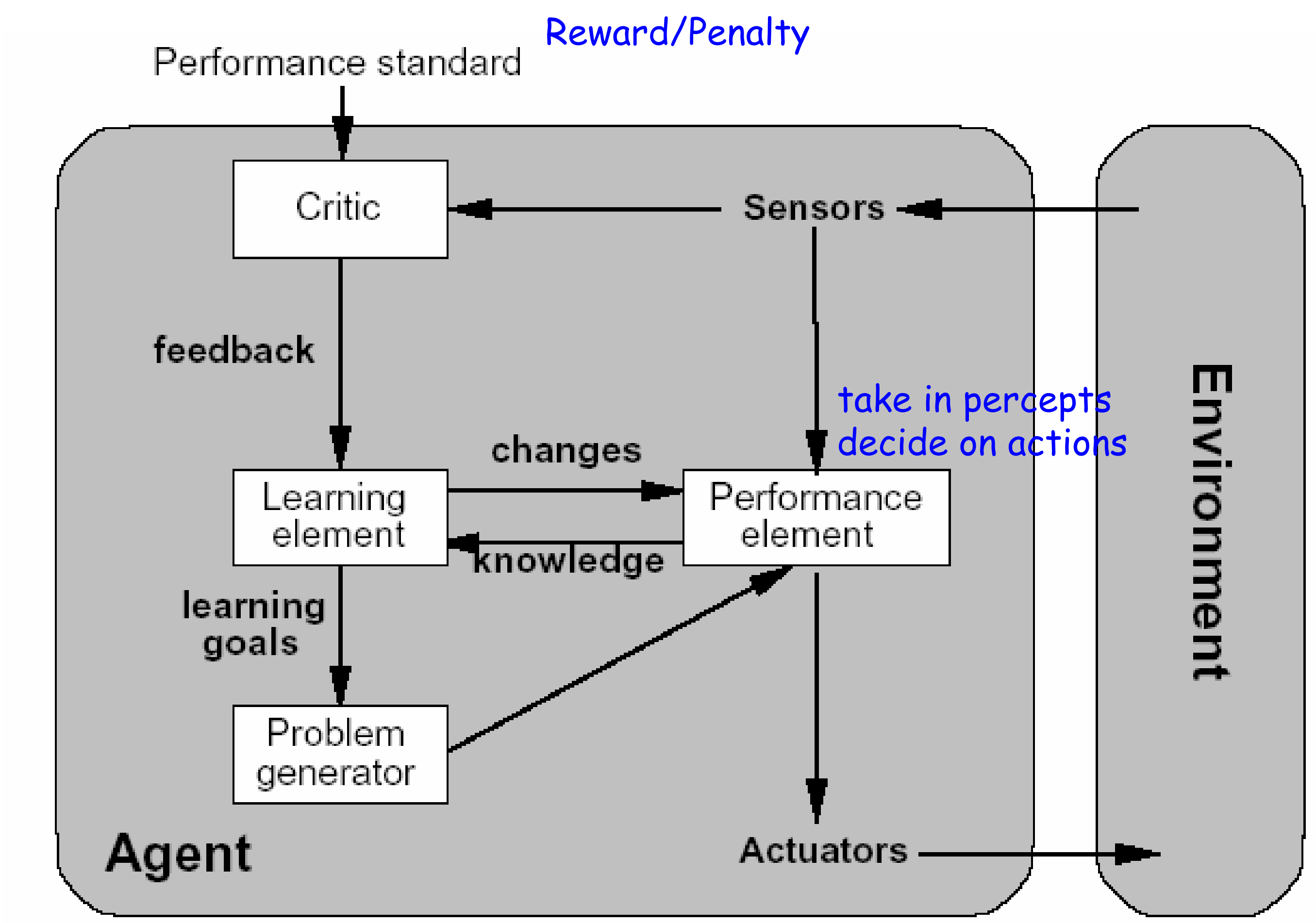


Learning Agents

- Learning allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge might allow
  - Learning algorithms
  - Create state-of-the-art agent!

- A learning agent composes of
  - **Learning element**: making improvements
  - **Performance element**: selecting external action
  - **Critic**: determining how the performance element should be modified according to the learning standard
    - Supervised/Unsupervised
  - **Problem generator**: suggesting actions that lead to new and informative experiences if the agent is willing to explore a little

Learning Agents (cont.)



Learning Agents (cont.)

- For example, the taxis-driver agent makes a quick left turn across three lines if traffic
  - The critic observes the shocking language from other drivers
  - And the learning element is able to formulate a rule saying this was a bad action
  - Then the performance element is modified by install the new rule
- Besides, the problem generator might identify certain areas if behavior in need of improvement and suggest experiments
  - Such as trying out the brakes on different road surface under different conditions