Chapter 2

Intelligent Agents

CS361 Artificial Intelligence

Dr. Khaled Wassif

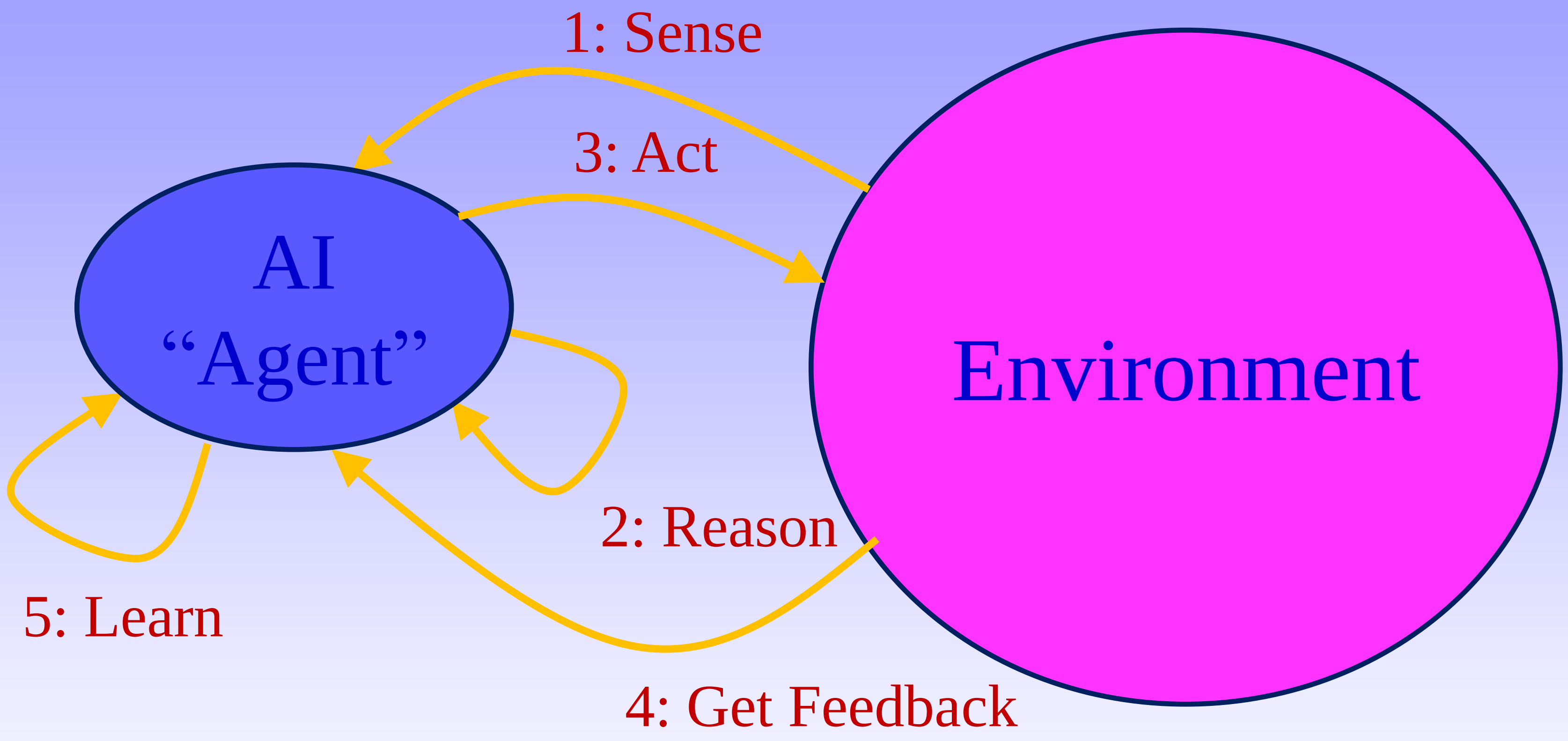
Spring 2022

(This is the instructor's notes and student has to read the textbook for complete material.)

Chapter Outline

- Agents and Environments
- How Agents Should Act
- Rational Agents
- Task Environment of Agent
- Structure of Agents

The Big AI Picture

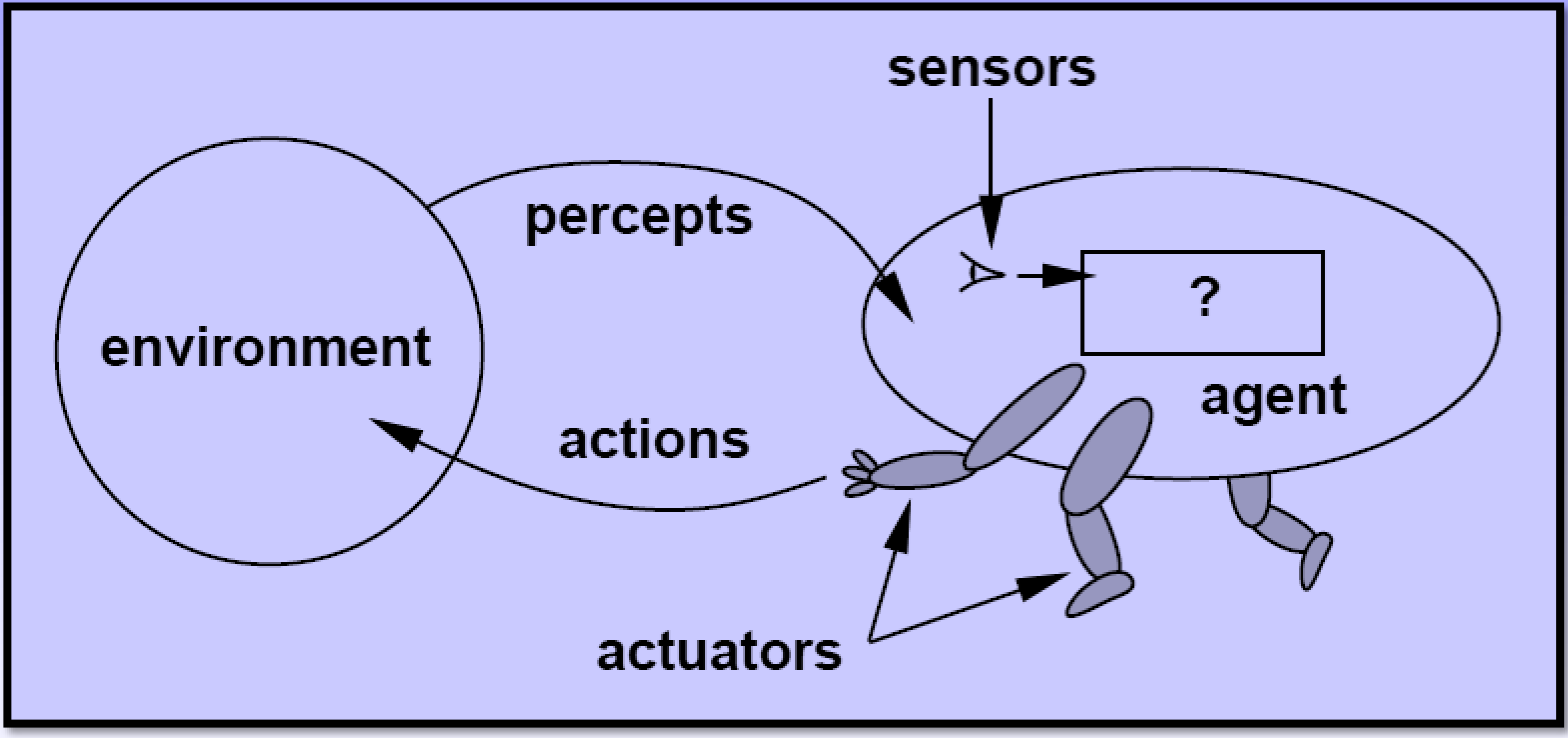


The study of 'agents' that exist in an environment and perceive, act, and learn

Agents and Environments

- An **agent** is anything that can be viewed as *perceiving* its environment through sensors and *acting* upon that environment through actuators.
- Human agent has:
  - Eyes, ears, and other organs for sensors;
  - Hands, legs, mouth, and other body parts for actuators.
- Robotic agent has:
  - Cameras and infrared range finders for sensors;
  - Various motors for actuators.
- Software agent:
  - Receives keystrokes, file contents, and network packets as inputs;
  - Displays on screen, writes files, and sends net packets as outputs.

Agents and Environments



Agents interact with environments through sensors and actuators

Agents and Environments

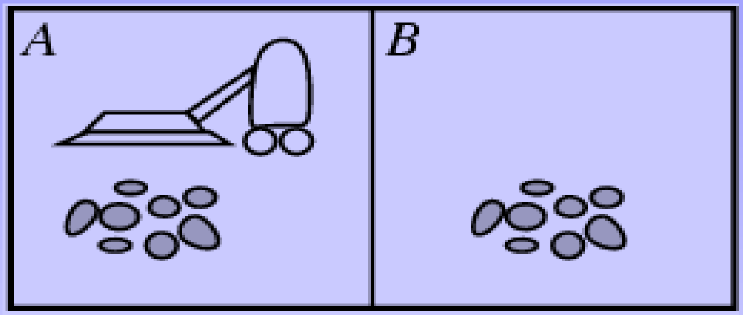
- An **agent's choice of action** at any given instant can depend on the entire percept sequence (history) observed to date.
- Mathematically, an agent's behavior is described by the agent function that maps from percept histories to actions:

$$[\ f: \mathcal{P}^{\star} \rightarrow \mathcal{A}\ ]$$

- Internally, the agent function for an artificial agent will be implemented by an agent program, running on the agent architecture to produce *f* :

agent = architecture + program

Vacuum-cleaner world



- *Environment*: just two locations – squares *A* and *B*
- *Percepts*: which square it is in (location) and whether there is dirt in the square (contents), e.g., [A, Dirty]
- *Actions*: move left, move right, suck up the dirt, or do nothing
- *Agent function*: if the current square is dirty, then suck; otherwise move to the other square.

How Agents Should Act

- What makes an agent good or bad, intelligent or stupid?
- A **rational agent** is one that does "***the right thing***", based on what it can perceive and the actions it can perform.
- The right action is the one that will cause the agent to be most successful
  - Leaves us with the problem of deciding ***how*** and ***when*** to evaluate the agent's success.
- We use the term **performance measure** for the how — the criteria that determine how successful an agent is.
- The when of evaluating performance is also important.

How Agents Should Act

- Obviously, there is not one fixed measure suitable for all agents.
  - There are different criteria for different agents.
- Therefore, we will insist on an *objective* performance measure imposed by some authority.
- *Outside observers* establish a standard of what it means to be successful in an environment and use it to measure the performance of agents.
  - E.g., performance measure of a vacuum-cleaner agent could be amount of dirt cleaned up, amount of time taken, amount of electricity consumed, amount of noise generated, etc.

Example: Coffee-Delivery Agent

- Operate in a suite of offices.
- Receives coffee from kitchen and delivers to offices.
- <u>How</u> can measure of the agent success?
  - … …
  - … …
- <u>When</u> to measure its success?
  - After the first attempt?
  - After the *n*th attempt?
  - At a fixed time, taking the average over all attempts so far?

Rationality vs. Success

- Note that: rational ≠ successful
- Rationality depends not only on the performance measure, but also on:
  - Agent's sensory capabilities (what it can sense).
    - » Coffee-agents that detect obstacles visually vs. those that use collision detection.
  - Agent's actuator capabilities (what it can do).
    - » Coffee-agents that can heat coffee with a built-in coil vs. those that cannot.
  - Agent's knowledge.
    - » Informed agents, uninformed agents, misinformed agents.

Rational Agents

- 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.
- **Rational Agent Definition:**

  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.

Rational Agents

- Need to distinguish between rationality and ***omniscience***
  - An omniscient agent (all-knowing with infinite knowledge) knows the actual outcome of its actions, and can act accordingly.
  - But omniscience is impossible in reality.
- Rationality is not the same as ***perfection***
  - Rationality maximizes expected performance, while perfection maximizes actual performance.
- Agents, to be rational, can perform actions to obtain useful information in order to modify future percepts (information gathering, exploration).
- A rational agent should be ***autonomous***
  - Its behavior is determined by its own experience (with ability to learn and adapt its prior knowledge).

Specifying the Task Environment

- To design an agent, the first step is to specify its *task environment* as fully as possible.
- The **task environment** of any agent includes:
  1. Performance measure.
  2. Environment.
  3. Actuators.
  4. Sensors.
- For the acronymically minded, we call this the **PEAS** (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors) description.

PEAS of Taxi Driver

- The task environment of designing an automated taxi driver:
  - Performance measure: Safe, fast, legal, comfortable trip, maximize profits.
  - Environment: Roads, other traffic, pedestrians, customers.
  - Actuators: Steering wheel, accelerator, brake, signal, horn.
  - Sensors: Cameras, sonar, speedometer, odometer, GPS, engine sensors, keyboard

PEAS of Diagnosis System

- The task environment of designing a medical diagnosis system:
  - Performance measure: Healthy patient, minimize costs, lawsuits.
  - Environment: Patient, hospital, staff.
  - Actuators: Screen display (questions, tests, diagnoses, treatments, referrals)
  - Sensors: Keyboard entry of symptoms, findings, patient's answers

PEAS of Part-picking Robot

- The task environment of designing a part-picking robot:
  - Performance measure: Percentage of parts in correct bins.
  - Environment: Conveyor belt with parts, bins.
  - Actuators: Jointed arm and hand
  - Sensors: Camera, joint angle sensors

PEAS of Interactive English Tutor

- The task environment of designing an interactive English tutor:
  - Performance measure: Maximize student's score on test.
  - Environment: Set of students, testing agency.
  - Actuators: Screen display of exercises, suggestions, corrections.
  - Sensors: Keyboard entry.

Properties of Task Environments

- The task environments can be categorized according to number of dimensions:
  - **Fully observable** (vs. **partially observable**): An agent's sensors give it access to the complete state of the environment at each point in time.
  - **Single agent** (vs. **multi-agent**): An agent operating by itself in an environment.
  - **Deterministic** (vs. **stochastic**): Next state of the environment is completely determined by the current state and the action executed by the agent.
  - **Episodic** (vs. **sequential**): In an episodic task environment, the agent's experience is divided into atomic episodes – each episode consists of the agent percept and then performing a single action.

Properties of Task Environments

- The task environments can be categorized according to number of dimensions:
  - **Dynamic** vs. **static**: If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise, it is static.
  - **Discrete** (vs. **continuous**): A limited number of distinct, clearly defined percepts and actions.
  - **Known** (vs. **unknown**): The rules of the game, or physics/dynamics of the environment are known to the agent.
- These properties of various task environments determine the appropriate agent design and the applicable technique for agent implementation.

Properties of Task Environments

- The task environments can be categorized according to number of dimensions:
  - **Fully observable** (vs. **partially observable**): An agent's sensors give it access to the complete state of the environment at each point in time.
    - » A task environment is effectively fully observable if the sensors detect all aspects that are *relevant* (depends on the performance measure) to the choice of action.
    - » 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:
    - » If the agent has no sensors at all then the environment is unobservable.

Properties of Task Environments

- The task environments can be categorized according to number of dimensions:
  - **Single agent** (vs. **multi-agent**): An agent operating by itself in an environment.
    - » 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.
    - » Chess is a **competitive** multi-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.
    - » In the taxi-driving environment, on the other hand, avoiding collisions maximizes the performance measure of all agents, so it is a partially **cooperative** multi-agent environment.

Properties of Task Environments

- The task environments can be categorized according to number of dimensions:
  - **Episodic** (vs. **sequential**): In an episodic task environment, the agent's experience is divided into atomic episodes – each episode consists of the agent percept and then performing a single action.
    - » Choice of action in each episode depends only on the episode itself.
      - Next episode does not depend on the actions taken in previous episodes.
    - » In sequential environments, 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

- The task environments can be categorized according to number of dimensions:
  - **Deterministic** (vs. **stochastic**): Next state of the environment is completely determined by the current state and the action executed by the agent.
    - » In principle, an agent need not worry about *uncertainty* in a fully observable, deterministic environment.
      - If the environment is deterministic except for the actions of other agents; in a multi-agent environment, then the environment is **strategic**.
      - 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; therefore, they must be treated as stochastic.
    - » It is often better to think of an environment as deterministic or stochastic *from the point of view of the agent*.

Properties of Task Environments

- The task environments can be categorized according to number of dimensions:
  - **Dynamic** vs. **static**: 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**.

Properties of Task Environments

- The task environments can be categorized according to number of dimensions:
  - **Discrete** (vs. **continuous**): A limited number of distinct, clearly defined percepts and actions.
    - » This 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:
      - Chess environment has a finite number of distinct states (excluding the clock) and 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, and 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

- The task environments can be categorized according to number of dimensions:
  - **Known** (vs. **unknown**): The rules of the game, or physics/dynamics of the environment are known to the agent.
    - » 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.
    - » The distinction between known and unknown environments is not the same as the one between fully and partially observable environments.
      - It is 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.

Properties of Task Environments

	Chess with a clock	Chess without a clock	Taxi driving
Fully observable	Yes	Yes	No
Single agent	No	No	No
Deterministic	Strategic	Strategic	No
Episodic	No	No	No
Static	Semi	Yes	No
Discrete	Yes	Yes	No
Known	Yes	Yes	Yes

- The hardest case is *partially observable*, *multi-agent*, *stochastic*, *sequential*, *dynamic*, *continuous*, and *unknown*.

Structure of Agents

- AI job is to design the agent program that implements the agent function mapping percepts to actions.
- Obviously, the designed program for an agent must to be appropriate for that agent architecture.

Agent = Architecture + Program

- The agent architecture is some sort of computing device, the program run on it, with physical sensors and actuators.
- The architecture has three functions:
    1. Makes the percepts from sensors available to the program.
    2. Runs the program.
    3. Feeds the generated program's action to the actuators.

Skeleton Agent

```
function SKELETON-AGENT (percept) returns action
    static: memory
    memory ← UPDATE-MEMORY(memory, percept)
    action ← CHOOSE-BEST-ACTION(memory)
    memory ← UPDATE-MEMORY(memory, action)
    return action
```

Skeleton Agent (cont.)

- On each invocation:
  - the agent's memory is updated to reflect the new percept
  - the best action is chosen
  - the fact that the action was taken is also stored in memory
- The memory persists from one invocation to the next.
- It is up to the agent, if it so desires, to build up the percept sequence in memory.
  - In some environments, it is possible to be quite successful without storing the percept sequence.
  - In complex domains, it is infeasible to store the complete sequence.
- The performance measure; not part of skeleton program; is applied externally to judge the behavior of the agent.

Table-Driven Agent

- Operates by keeping in memory its percept sequence, and using it to index into ***table***, which contains the appropriate action for all possible percept sequences.
- Why this proposal is doomed to failure?
  1. The table needed for something as simple as an agent that can only play chess would be about 35^{100} entries.
  2. The designer will take quite a long time to build the table.
  3. The agent has no autonomy at all, if the environment changed in some unexpected way, the agent would be lost.
  4. Even if the agent has a learning mechanism as well, to have a degree of autonomy, it would take forever to learn the right table entries.
- **AI challenge** is to find out how to write a program that produce rational behavior from a small amount of code rather than from a large number of table entries.

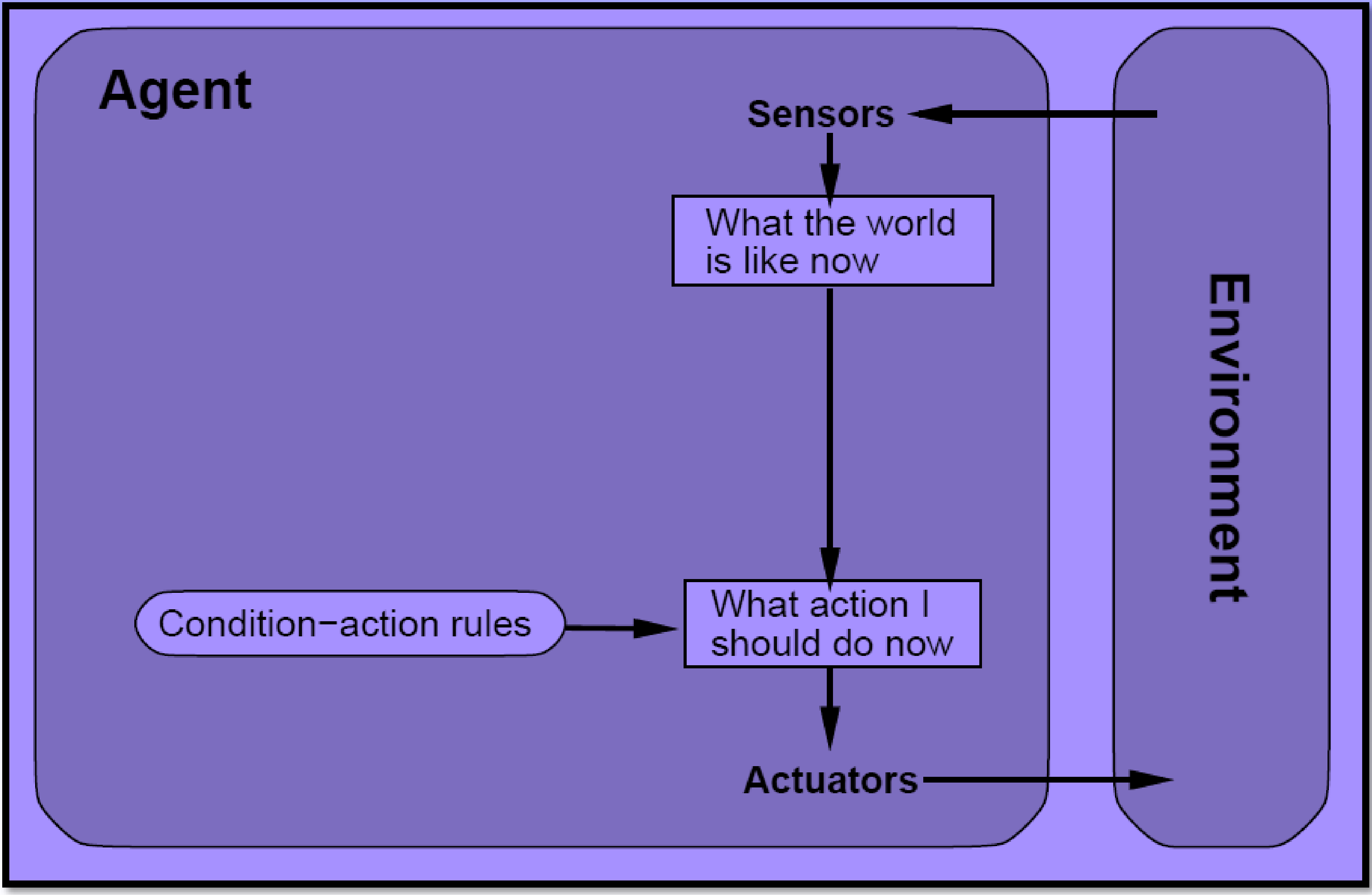
Table-Driven Agent (cont.)

```
function TABLE-DRIVEN-AGENT (percept) returns 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 ← LOOK-UP(table, percepts)

    return action
```

Agent Program

- We will consider four types of agent program that represent the principles underlying of almost all intelligence systems:
  - Simple reflex agents
  - Model-based reflex agents
  - Goal-based agents
  - Utility-based agents

Simple Reflex Agents



Simple Reflex Agents

- Simplest kind of agents.
- These agents select actions on the basis of the current percept, ignoring the rest of the percept history.
- The agent structure showing how condition-action rules allow making the connection from percept to action.
- The agent program is very simple:
  - The INTERPRET-INPUT function generates an abstracted description of the current state from the percept.
  - The RULE-MATCH function returns the first rule in the set of rules that matches the given state description.
- Work only if the environment is fully observable.
  - A little bit of observability can cause serious trouble.

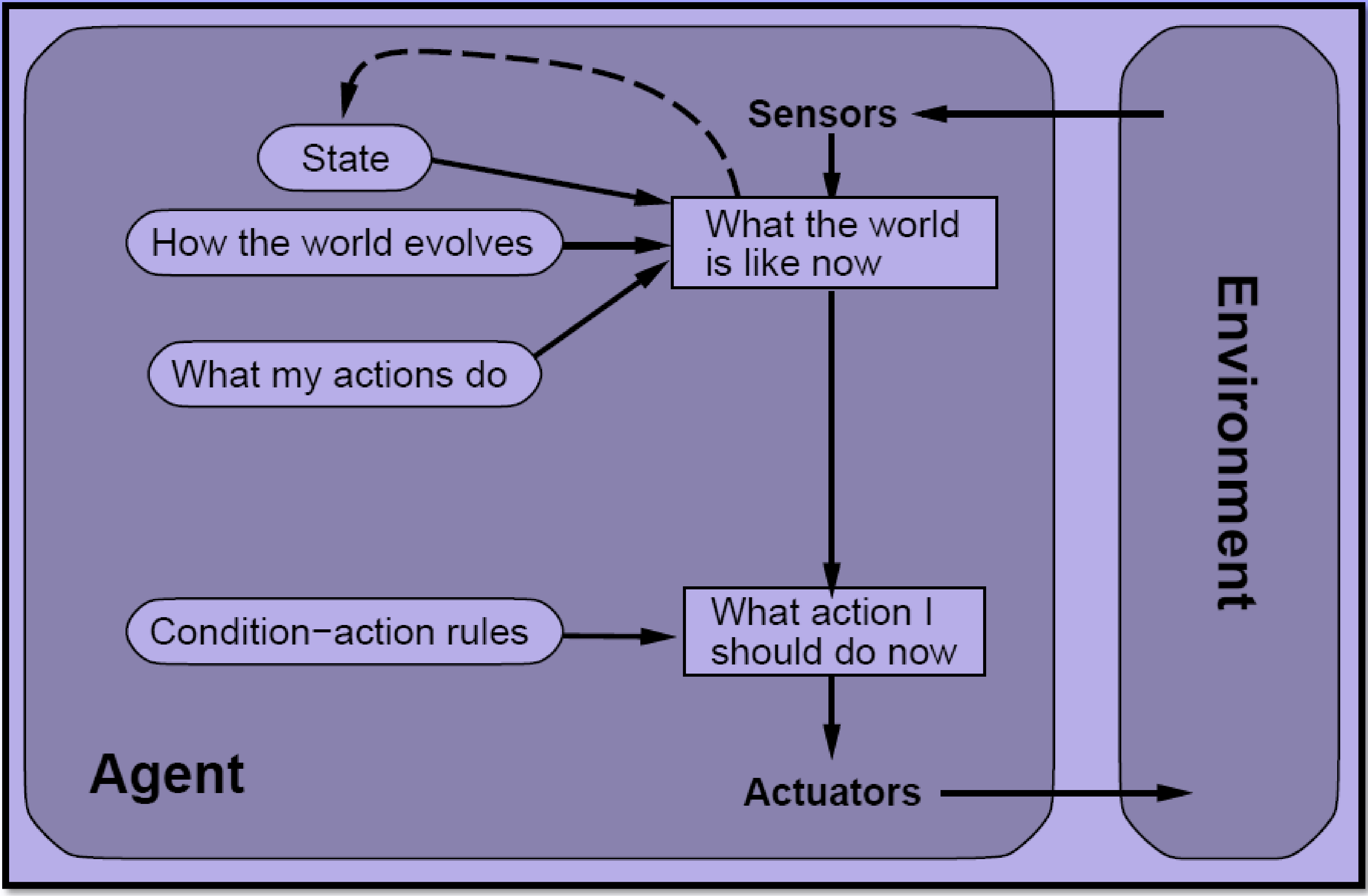
Simple Reflex Agents

```
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
```

Model-based Reflex Agents

- Most effective way to handle partial observability is to keep track of part of the world it can not see now.
  - The agent maintains some sort of internal state that depends on percept history and reflects some aspects of the current state.
- Two kinds of knowledge, which model the world, must be encoded in the agent program to update the internal state information:
  - First, we need some information about how the world evolves independently of the agent.
  - Second, we need some information about how the agent's own actions affect the world.
- The agent structure showing how the current percept is combined with the old internal state to generate the updated description of the current state.

Model-based Reflex Agents



Model-based Reflex Agents

```
function MODEL-BASED-REFLEX-AGENT(percept) returns action
    static: 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[rule]
    return action
```

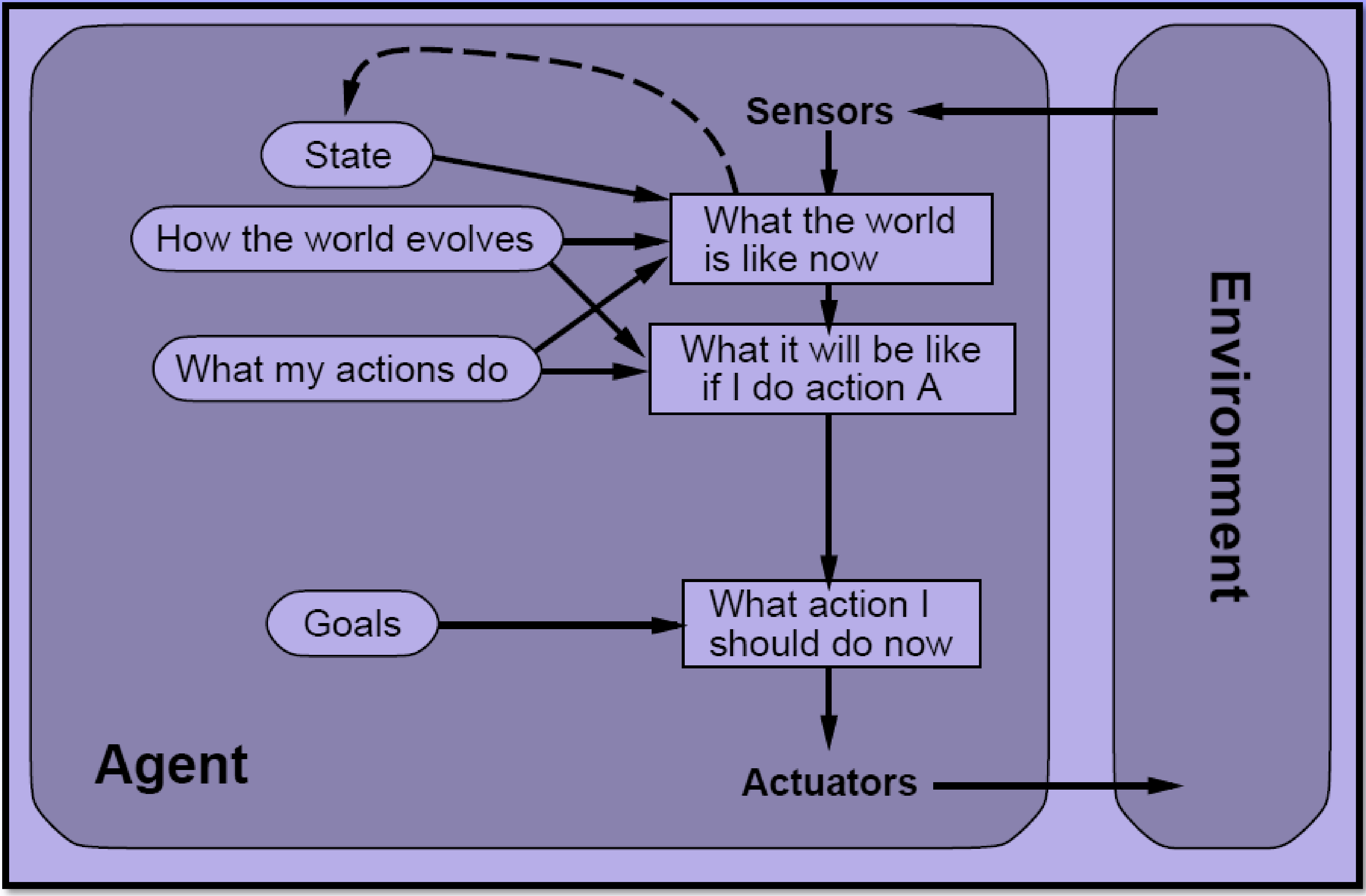
Model-based Reflex Agents

- 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.
- Regardless of the kind of representation used, it is seldom possible for the agent to determine *exactly* the current state of a partially observable environment.
  - For that reason, **uncertainty** about the current state may be unavoidable, but the agent still has to make a decision.
  - Therefore, the internal "state" maintained by a model-based agent may be not describe "what the world is like now" in a literal sense.

Goal-based Agents

- Knowing about the current state of the environment is not always enough to decide what to do.
  - The agent needs some sort of **goal** information, which describes situations that are desirable.
- The agent program can combine the goal with the model (same information used in the model-based reflex agent) in order to choose actions that achieve that goal.
  - Sometimes, goal-based action will be simple, when goal satisfaction results immediately from a single action.
  - Sometimes, it will be more tricky, when the agent has to consider many trials to find a way to achieve the goal.

Goal-based Agents


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

Goal-based Agents

```
function GOAL-BASED-AGENT (percept) returns action
    static: state, a description of the current world state
            rules, a set of condition-action rules
            action, the most recent action, initially none
            goal, the goal trying to achieve
    state ← UPDATE-STATE(state, action, percept, model)
    rules ← RULE-MATCH(state, rules)
    actions ← RULE-ACTION[rules]
    action ← CHOOSE-ACTION(goal, actions)
    return action
```

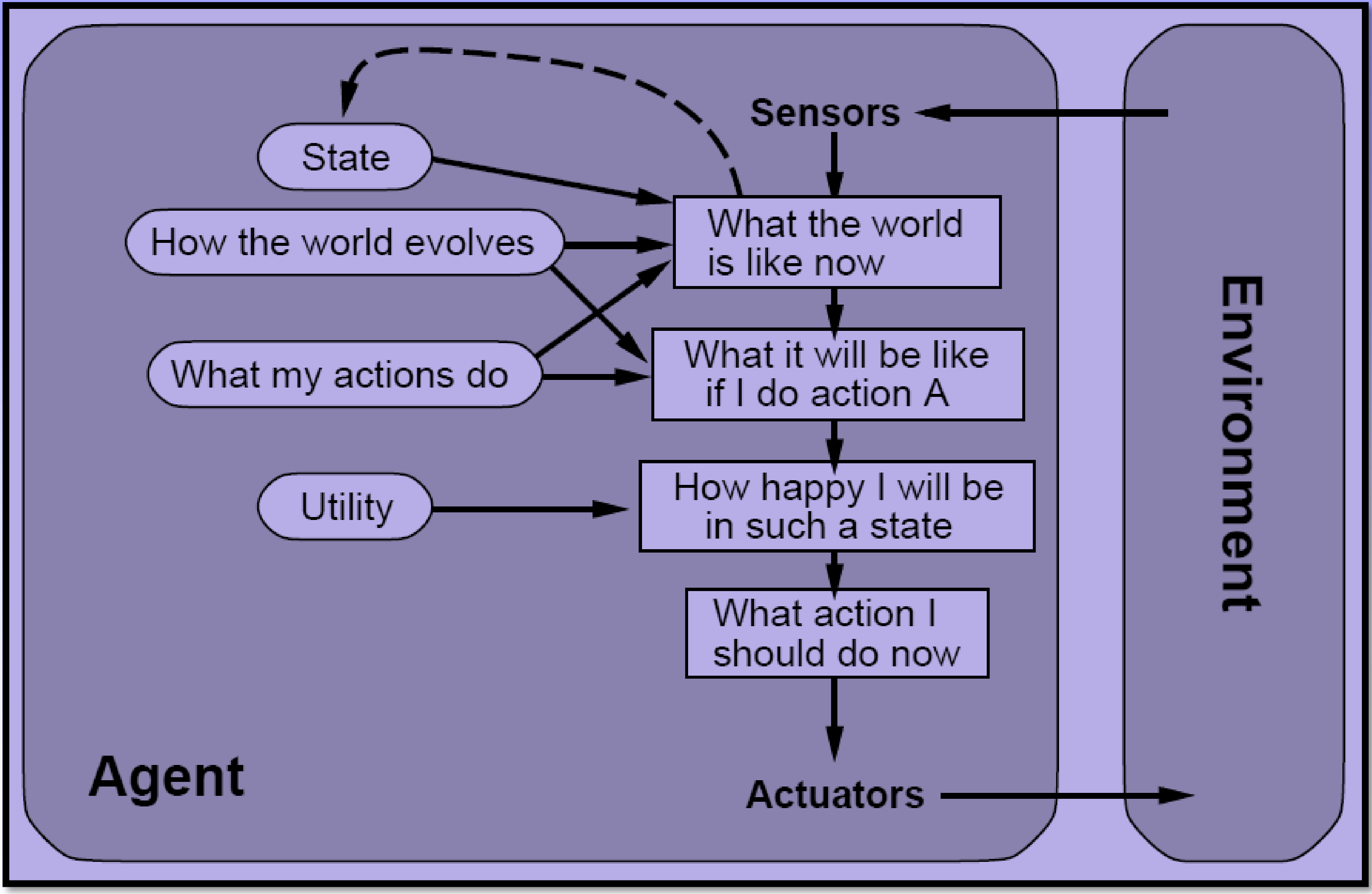
Goal-based Agents

- **Search** and **planning** are the subfields of AI devoted to finding action sequences that achieve the agent's goals.
- Decision making of this kind is different from the condition–action rules described earlier.
  - It involves consideration of the future for both:
    - » What will happen if I do such-and-such?
    - » Will that make me happy?
- This agent is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified.

Utility-based Agents

- Goals alone are not really enough to generate high-quality behavior in most environment.
  - Goals just provide a crude binary distinction between "happy" and "unhappy" states.
  - Need a comparison of different world states according to exactly how happy they would make the agent.
- Therefore, the agent wants some sort of utility function that internally measures its performance among different states of the world.
  - 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

Utility-based Agents


Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action A
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

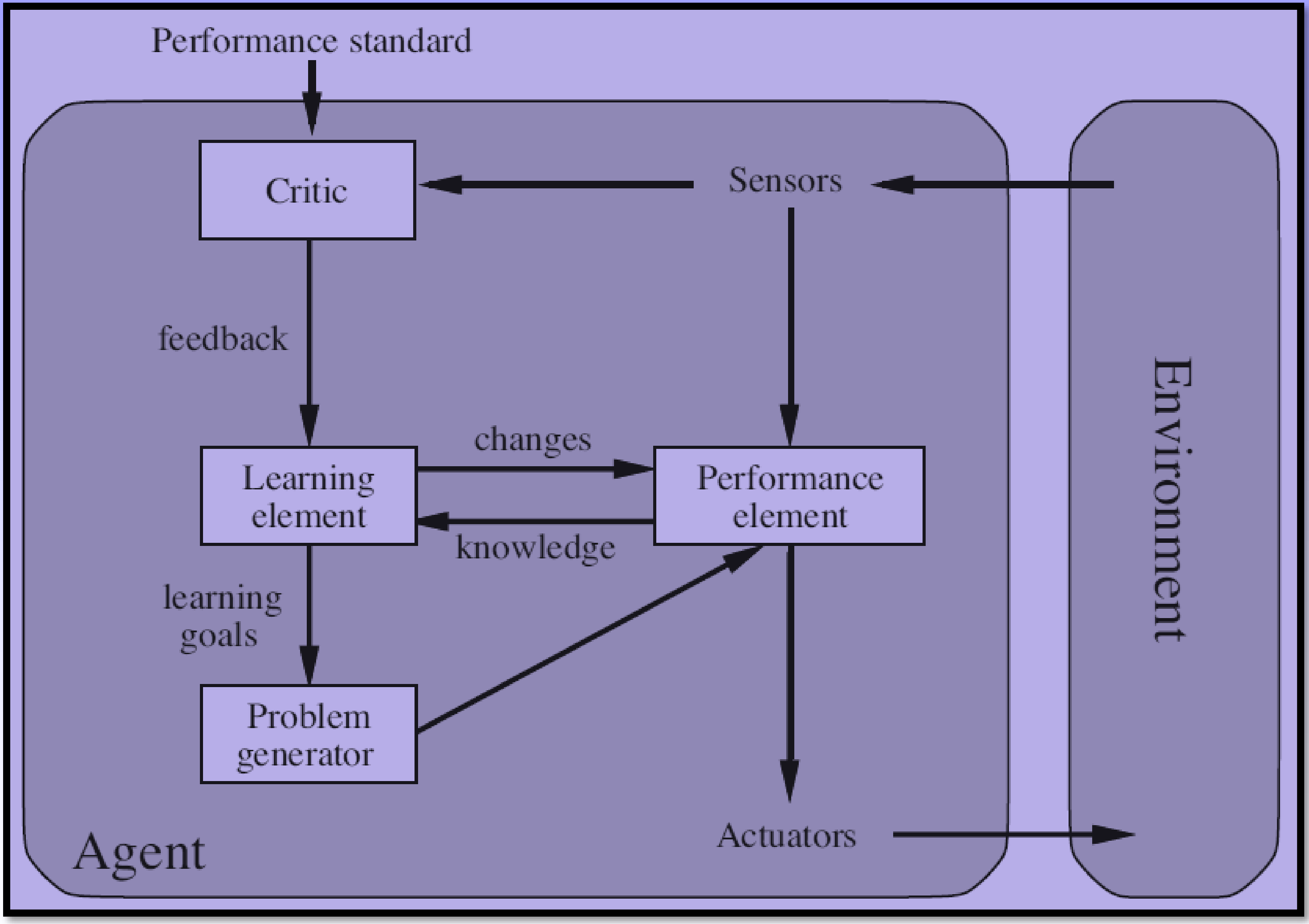
Utility-based Agents

```
function UTILITY-BASED-AGENT (percept) returns action
    static: state, a description of the current world state
            rules, a set of condition-action rules
            action, the most recent action, initially none
            utility, the function that measures its performances
    state ← UPDATE-STATE(state, action, percept, model)
    rules ← RULE-MATCH(state, rules)
    actions ← RULE-ACTION[rules]
    action ← CHOOSE-ACTION(utility, actions)
    return action
```

Utility-based Agents

- In two kinds of cases, goals are inadequate but a utility-based agent can still make rational decisions:
  - First, when there are *conflicting goals*, only some of which can be achieved,
    - » The utility function specifies the appropriate trade-off.
  - Second, when there are *several goals* that the agent can aim for, none of which can be achieved with certainty,
    - » Utility provides a way in which the likelihood of success can be weighed up against the importance of the goals.
- Technically, a rational utility-based agent chooses the action that maximizes the **expected utility** of the action outcomes.

Learning Agents



Summary

- Agents interact with environments through actuators and sensors.
- Agent function specifies the action taken by the agent in response to any percept sequence.
- Performance measure evaluates the environment sequence.
- A 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:
  - simple reflex, model-based, goal-based, utility-based
- All agents can improve their performance through learning.