
Intelligent Agents

Fundamentals of Artificial Intelligence

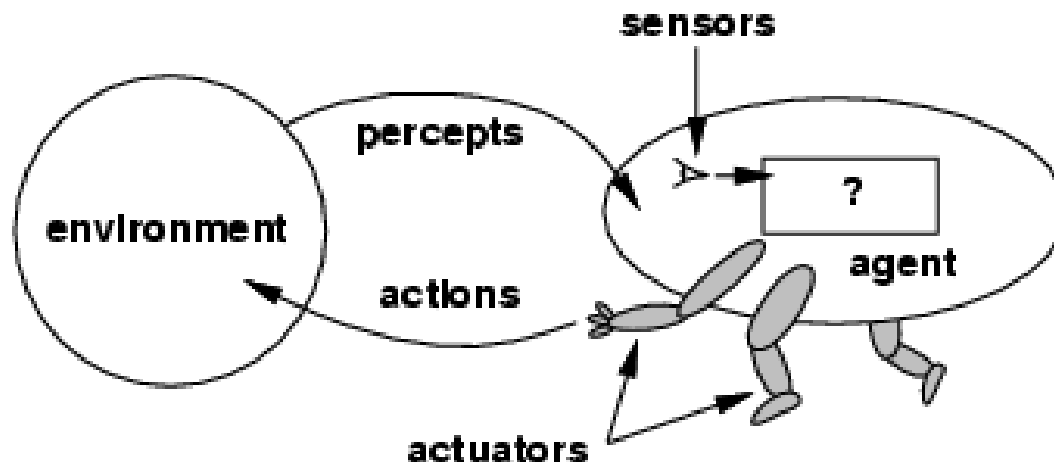
Slides are mostly adapted from AIMA and Svetlana Lazebnik (UIUC)

Outline

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

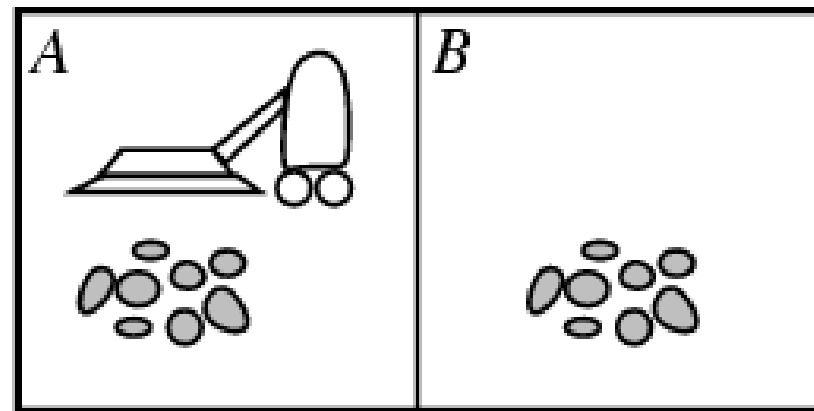
Agents

- An **agent** is anything that can be viewed as
 - **perceiving** its **environment** through **sensors** and
 - **acting** upon that environment through **actuators**
 - Assumption: Every agent can perceive its own actions (but not always the effects)



Example: Vacuum-Agent

- **Percepts:**
Location and status,
e.g., [A,Dirty]
- **Actions:**
Left, Right, Suck, NoOp

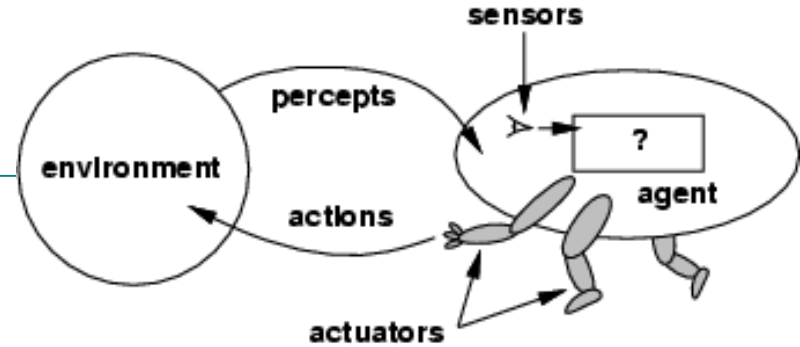


- function Vacuum-Agent([location,status])** returns an **action**
- *if status = Dirty then return Suck*
 - *else if location = A then return Right*
 - *else if location = B then return Left*

Agents

- Human agent:
 - eyes, ears, and other organs for sensors;
 - hands, legs, mouth, and other body parts for actuators
 - Robotic agent:
 - cameras and infrared range finders for sensors;
 - various motors for actuators
 - A software agent:
 - Keystrokes, file contents, received network packages as sensors
 - Displays on the screen, files, sent network packets as actuators
-

Agents and environments



- **Percept**: agent's perceptual input at any given instant
- **Percept sequence**: complete history of everything the agent has ever perceived
- An agent's choice of action at any given instant can depend on the entire percept sequence observed to date
- An agent's behavior is described by the **agent function** which maps from percept histories to actions:
$$[f: \mathcal{P}^* \rightarrow \mathcal{A}]$$
- We can imagine tabulating the agent function that describes any given agent (External characterization)
- Internally, the agent function will be implemented by an **agent program** which runs on the physical **architecture** to produce f
- agent = architecture + program

Rational agents

- An agent should strive to "do 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
 - Performance measure: An objective criterion for success of an agent's behavior
 - 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.
 - As a general rule, it is better to design performance measures according to what one actually wants in the environment. Rather than according to how one thinks the agent should behave (amount of dirt cleaned vs a clean floor)
 - A more suitable measure would reward the agent for having a clean floor
-

Rationality

- 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 agents

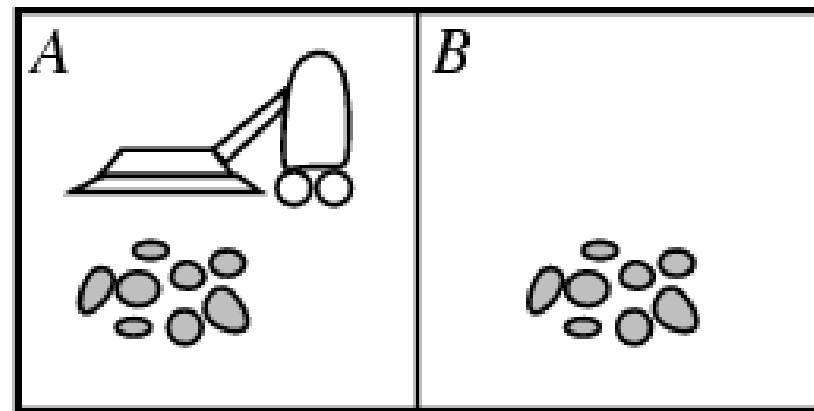
- For each possible percept sequence, a **rational agent** should select an action that is expected to maximize its **performance measure**, *given the evidence provided by the percept sequence and the agent's built-in knowledge*
 - **Performance measure (utility function):**
An *objective* criterion for success of an agent's behavior
 - **Expected utility:**
 - Can a rational agent make mistakes?

$$EU(action) = \sum_{outcomes} P(outcome | action)U(outcome)$$

outcomes
-

Back to Vacuum-Agent

- **Percepts:**
Location and status,
e.g., [A,Dirty]
- **Actions:**
Left, Right, Suck, NoOp



function 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*
- Is this agent rational?
 - Depends on performance measure, environment properties

Vacuum cleaner agent

- Let's assume the following
 - The performance measure awards one point for each clean square at each time step, over a lifetime of 1000 time steps
 - The geography of the environment is known a priori but the dirt distribution and the initial location of the agent are not. Clean squares stay clean and the sucking cleans the current square. The Left and Right actions move the agent left and right except when this would take the agent outside the environment, in which case the agent remains where it is
 - The only available actions are Left, Right, Suck and NoOp
 - The agent correctly perceives its location and whether that location contains dirt
 - Under these circumstances the agent is rational: its expected performance is at least as high as any other agent's
-

Vacuum cleaner agent

- Same agent would be irrational under different circumstances
 - once all dirt is cleaned up it will oscillate needlessly back and forth.
 - If the performance measure includes a penalty of one point for each movement left or right, the agent will fare poorly.
 - A better agent for this case would do nothing once it is sure that all the squares are clean.
 - If the clean squares can become dirty again, the agent should occasionally check and clean them if needed.
 - If the geography of the environment is unknown the agent will need to explore it rather than stick to squares A and B
-

Rational agents

- Rationality is distinct from omniscience (all-knowing with infinite knowledge)
 - Rationality maximizes expected performance while perfection maximizes actual performance
 - Agents can perform actions in order to modify future percepts so as to obtain useful information (information gathering, exploration)
 - An agent is **autonomous** if its behavior is determined by its own experience (with ability to learn and adapt)
-

Specifying the task environment (PEAS)

- PEAS:
 - Performance measure,
 - Environment,
 - Actuators,
 - Sensors
 - In designing an agent, the first step must always be to specify the task environment (PEAS) as fully as possible
-

Specifying the task environment

- **PEAS**: Performance measure, Environment, Actuators, Sensors
 - **P**: a function the agent is maximizing (or minimizing)
 - Assumed given
 - In practice, needs to be computed somewhere
 - **E**: a formal representation for *world states*
 - For concreteness, a tuple $(var_1=val_1, var_2=val_2, \dots, var_n=val_n)$
 - **A**: actions that change the state according to a *transition model*
 - Given a state and action, what is the successor state (or distribution over successor states)?
 - **S**: observations that allow the agent to infer the world state
 - Often come in very different form than the state itself
 - E.g., in tracking, observations may be pixels and state variables 3D coordinates
-

PEAS Example: Autonomous taxi

- **Performance measure**
 - Safe, fast, legal, comfortable trip, maximize profits
 - **Environment**
 - Roads, other traffic, pedestrians, customers
 - **Actuators**
 - Steering wheel, accelerator, brake, signal, horn
 - **Sensors**
 - Cameras, LIDAR, speedometer, GPS, odometer, engine sensors, keyboard
-

Another PEAS example: Spam filter

- **Performance measure**
 - Minimizing false positives, false negatives
 - **Environment**
 - A user's email account, email server
 - **Actuators**
 - Mark as spam, delete, etc.
 - **Sensors**
 - Incoming messages, other information about user's account
-

PEAS for 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 for a satellite image analysis system

- **Performance measure:** correct image categorization
 - **Environment:** downlink from orbiting satellite
 - **Actuators:** display categorization of scene
 - **Sensors:** color pixel arrays
-

PEAS for 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 for a refinery controller

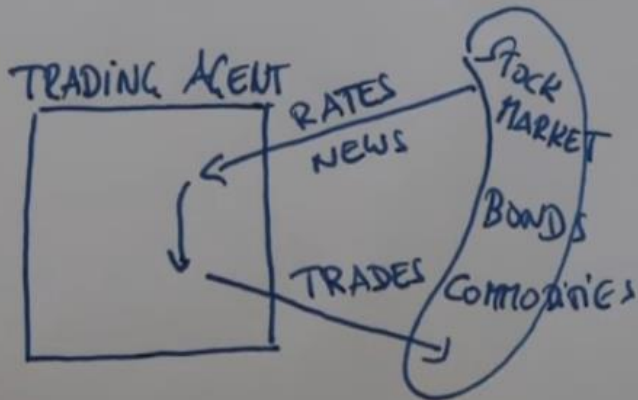
- **Performance measure:** maximize purity, yield, safety
 - **Environment:** refinery, operators
 - **Actuators:** valves, pumps, heaters, displays
 - **Sensors:** temperature, pressure, chemical sensors
-

PEAS for Interactive English tutor

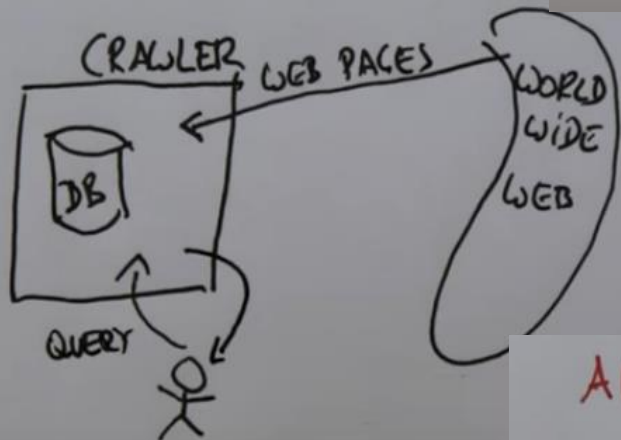
- **Performance measure:** Maximize student's score on test
 - **Environment:** Set of students
 - **Actuators:** Screen display (exercises, suggestions, corrections)
 - **Sensors:** Keyboard
-

Applications

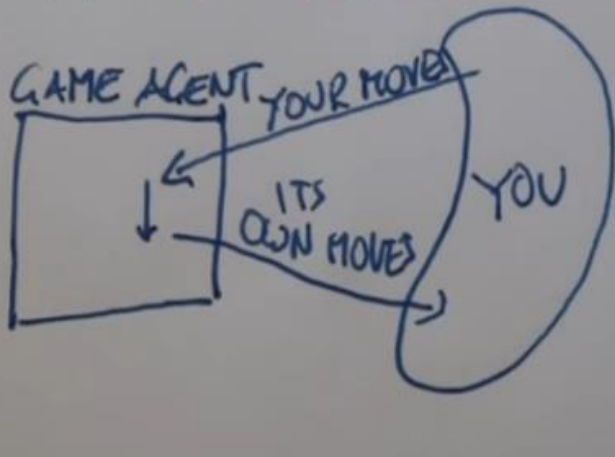
AI IN FINANCE



AI ON THE WEB



AI IN GAMES



AI IN ROBOTICS



AI IN MEDICINE

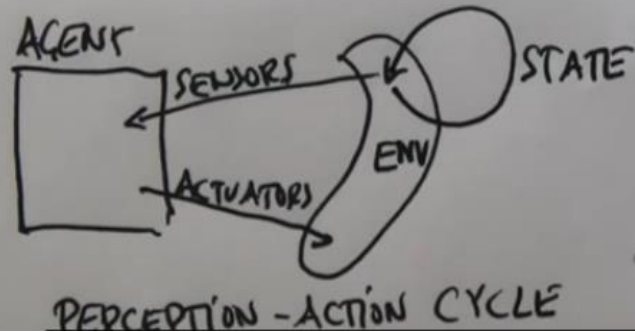


<https://classroom.udacity.com/courses/cs271/lessons/48688925/concepts/>

Environment types

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

FULLY versus PARTIALLY OBSERVABLE



Fully observable vs. partially observable:

- An environment is fully observable if an agent's sensors give it access to the complete state of the environment at each point in time.
- 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
- Examples: vacuum cleaner with local dirt sensor, taxi driver

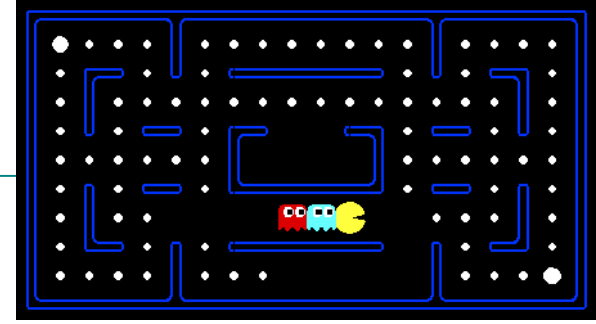
Environment types



Deterministic vs. stochastic:

- The environment is deterministic if the 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 partially observable then it could appear to be stochastic
 - Examples: Vacuum world is deterministic while taxi driver is not
 - If the environment is deterministic except for the actions of other agents, then the environment is **strategic**
-

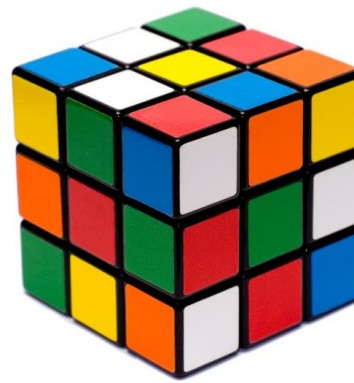
Environment types



Episodic vs. sequential:

- In episodic environments, the agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself.
- Examples: classification tasks
- In sequential environments, the current decision could affect all future decisions
- Examples: chess and taxi driver

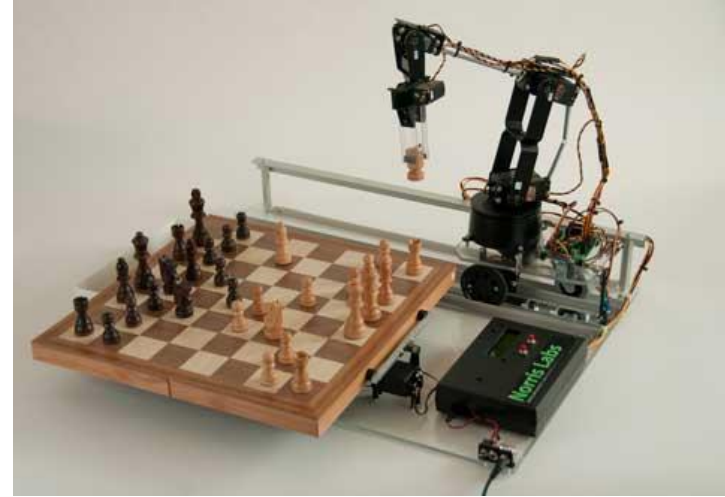
Environment types



Static vs. dynamic:

- The environment is unchanged while an agent is deliberating.
 - Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on the action or need it worry about the passage of time
 - Dynamic environments continuously ask the agent what it wants to do
 - The environment is **semi-dynamic** if the environment itself does not change with the passage of time but the agent's performance score does
 - Examples: taxi driving is dynamic, chess when played with a clock is semi-dynamic, crossword puzzles are static
-

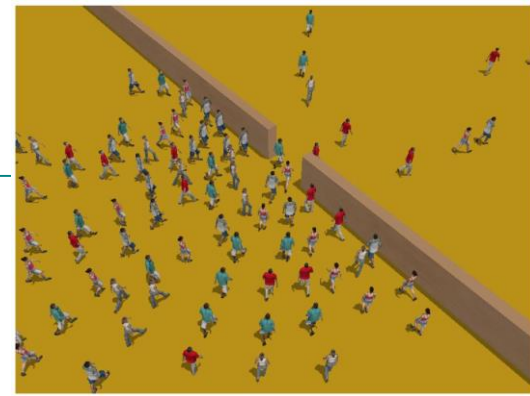
Environment types



Discrete vs. continuous:

- A limited number of distinct, clearly defined states, percepts and actions.
- Examples: Chess has finite number of discrete states, and has discrete set of percepts and actions. Taxi driving has continuous states, and actions

Environment types



Single agent vs. multiagent:

- An agent operating by itself in an environment is single agent
 - Examples: Crossword is a single agent while chess is two-agents
 - Question: Does an agent A have to treat an object B as an agent or can it be treated as a stochastically behaving object
 - Whether B's behaviour is best described by as maximizing a performance measure whose value depends on agent's A behaviour
 - Examples: chess is a competitive multiagent environment while taxi driving is a partially cooperative multiagent environment
-

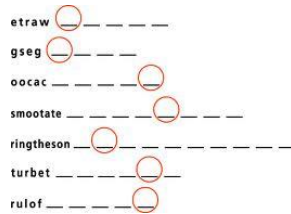
CHECKERS



FULLY OBSERVABLE
DETERMINISTIC
DISCRETE
ADVERSARIAL

	PARTIALLY OBSERVABLE	STOCHASTIC	CONTINUOUS	ADVERSARIAL
CHECKERS	O	O	O	
POKES			O	
ROBOT CAR				O

Examples of different environments



Word jumble solver



Chess with a clock



Scrabble



Autonomous driving

Observable

Fully

Fully

Partially

Partially

Deterministic

Deterministic

Strategic

Stochastic

Stochastic

Episodic

Episodic

Sequential

Sequential

Sequential

Static

Static

Semidynamic

Static

Dynamic

Discrete

Discrete

Discrete

Discrete

Continuous

Single agent

Single

Multi

Multi

Multi

Preview of the course

- **Deterministic environments:** search, constraint satisfaction, logic
 - Can be **sequential** or **episodic**
 - **Multi-agent, strategic environments:** minimax search, games
 - Can also be **stochastic, partially observable**
 - **Stochastic environments**
 - **Episodic:** Bayesian networks, pattern classifiers
 - **Sequential, known:** Markov decision processes
 - **Sequential, unknown:** reinforcement learning
-

Agent functions and programs

- An agent is completely specified by the agent function mapping percept sequences to actions
- One agent function (or a small equivalence class) is rational
- Aim: find a way to implement the rational agent function concisely -> design an **agent program**

Agent = agent program + architecture

- Architecture: some sort of computing device with physical sensors and actuators (PC, robotic car)
 - should be appropriate: walk action requires legs
-

Agent functions and programs

- Agent program:
 - Takes the current percept as input from the sensors
 - Return an action to the actuators
- While agent function takes the whole percept history, agent program takes just the current percept as input which the only available input from the environment
- The agent need to remember the whole percept sequence, if it needs it

Table-lookup agent

- A trivial agent program: keeps track of the percept sequence and then uses it to index into a table of actions to decide what to do
- The designers must construct the table that contains the appropriate action for every possible percept sequence

```

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
  
```

- Drawbacks:
 - Huge table (P^T , P: set of possible percepts, T: lifetime)
 - Space to store the table
 - Take a long time to build the table
 - No autonomy
 - Even with learning, need a long time to learn the table entries
-

Agent types

- Rather than a table how we can produce rational behavior from a small amount of code
 - Four basic types in order of increasing generality:
 - Simple reflex agents
 - Model-based reflex agents
 - Goal-based agents
 - Utility-based agents
-

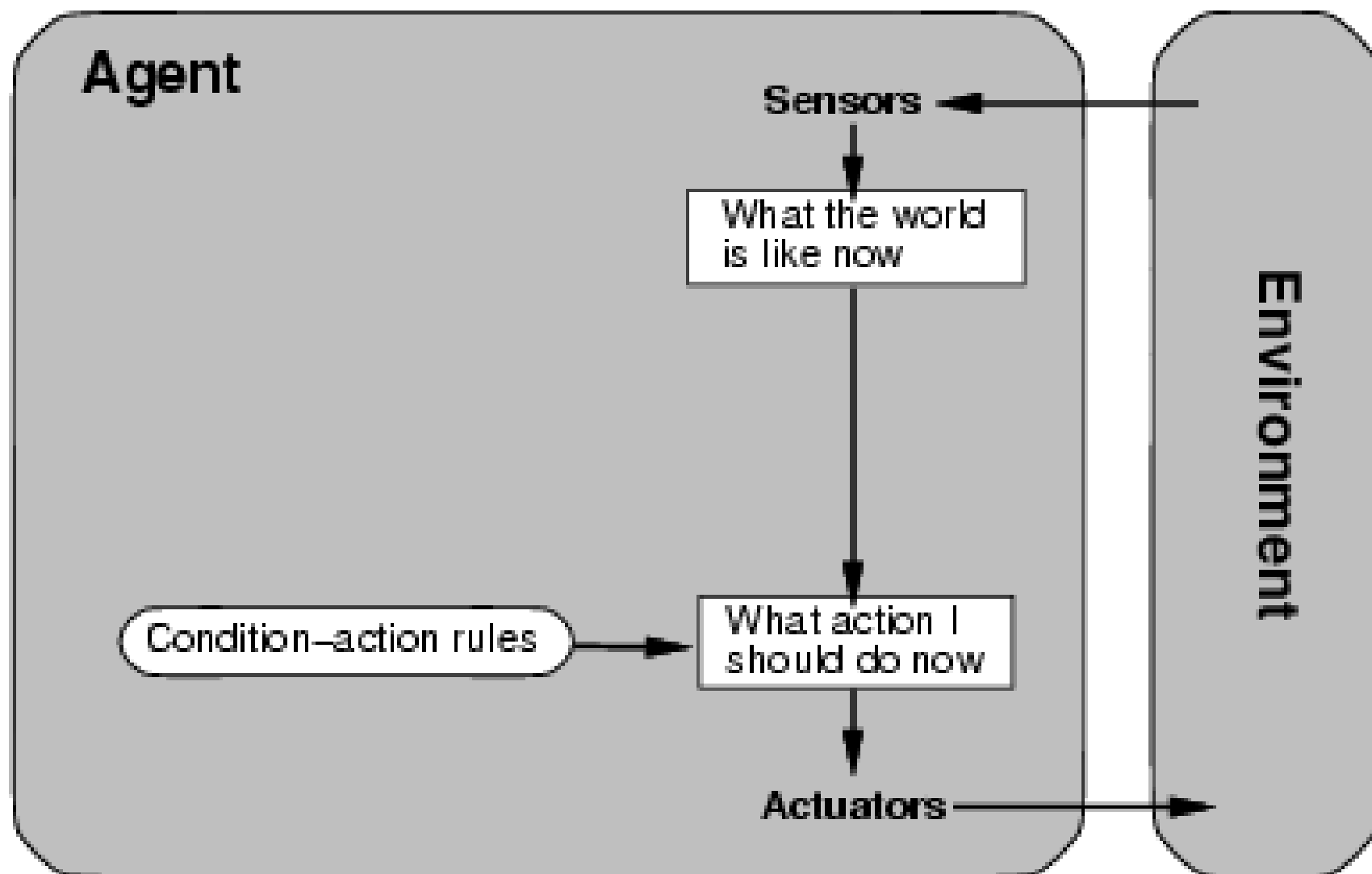
Simple reflex agents

- Select actions on the basis of the **current percept** ignoring the rest of the percept history
- Example: simple reflex vacuum cleaner agent

```
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
```

- **Condition-action-rule**
 - Example: **if** *car-in-front-is-breaking* **then** *initiate-breaking*
-

Simple reflex agents



Simple reflex agents

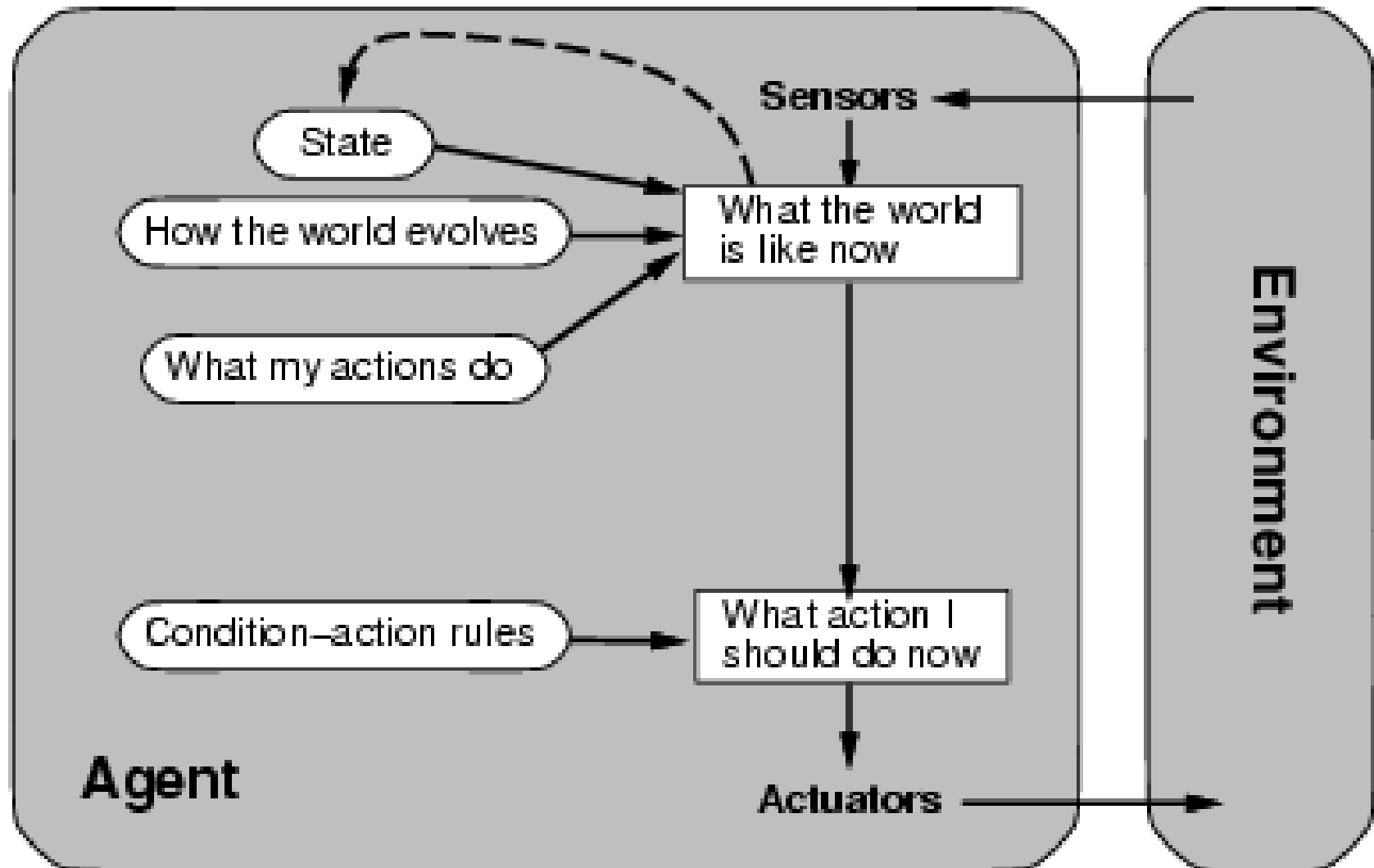
```
function SIMPLE-REFLEX-AGENT(percept) returns an action
static: rules, a set of condition-action rules
state <-- INTERPRET_INPUT(percept)
rule <-- RULE_MATCH(state, rules)
action <-- RULE_ACTION[rule]
return action
```

- Simple-reflex agents are simple, but they turn out to be of very limited intelligence
 - The agent will work only if the correct decision can be made on the basis of the current percept – that is only if the environment is fully observable
 - Infinite loops are often unavoidable – escape could be possible by randomizing
-

Model-based reflex agents

- The agent should keep track of the part of the world it can't see now
 - The agent should maintain some sort of internal state that depends on the percept history and reflects at least some of the unobserved aspects of the current state
 - Updating the internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program
 - Information about how the world evolves independently of the agent
 - Information about how the agent's own actions affects the world
 - Model of the world – model based agents
-

Model-based reflex agents



Model-based reflex agents

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_INPUT(*state*, *action*, *percept*)

rule <-- RULE_MATCH(*state*, *rules*)

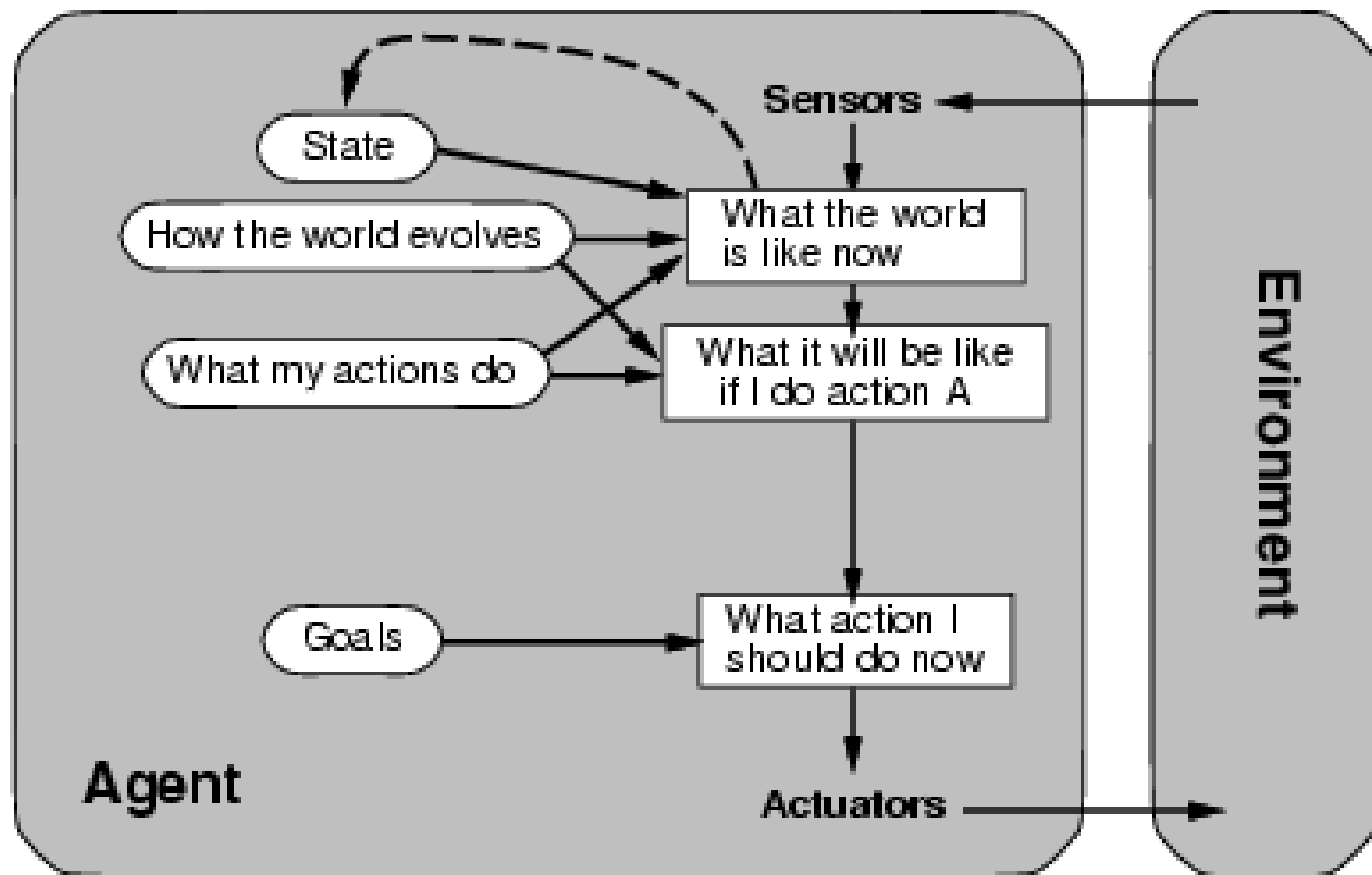
action <-- RULE_ACTION[*rule*]

return *action*

Goal-based agents

- Knowing about the current state of the environment is not always enough to decide what to do (e.g. decision at a road junction)
 - The agent needs some sort of goal information that describes situations that are desirable
 - The agent program can combine this with information about the results of possible actions in order to choose actions that achieve the goal
 - Usually requires search and planning
-

Goal-based agents



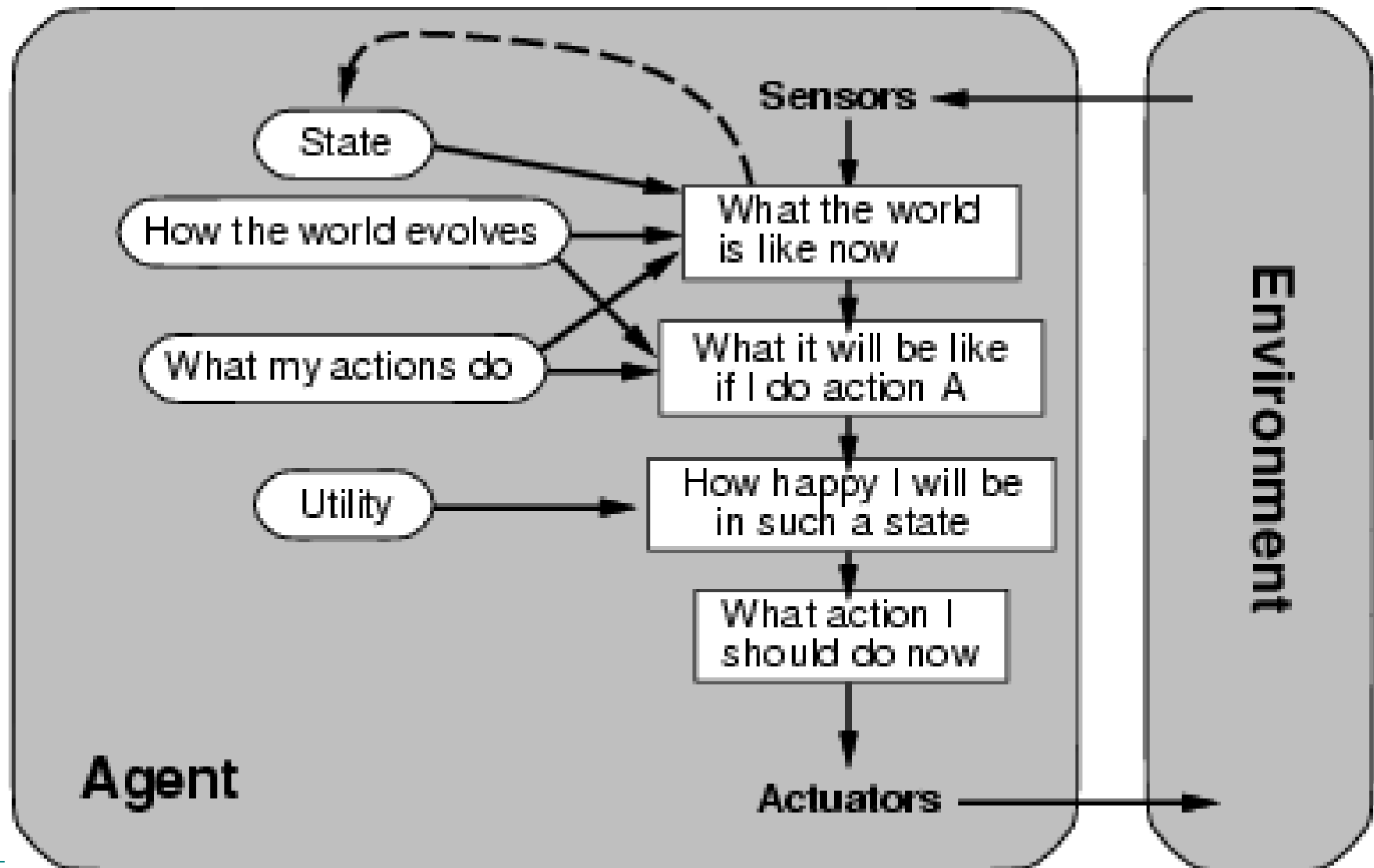
Goal-based agents vs reflex-based agents

- Although goal-based agents appears less efficient, it is more flexible because the knowledge that supports its decision is represented explicitly and can be modified
 - On the other hand, for the reflex-agent, we would have to rewrite many condition-action rules
 - The goal based agent's behavior can easily be changed
 - The reflex agent's rules must be changed for a new situation
-

Utility-based agents

- Goals alone are not really enough to generate high quality behavior in most environments – they just provide a 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 if they could be achieved
 - Happy – Utility (the quality of being useful)
 - A utility function maps a state onto a real number which describes the associated degree of happiness
-

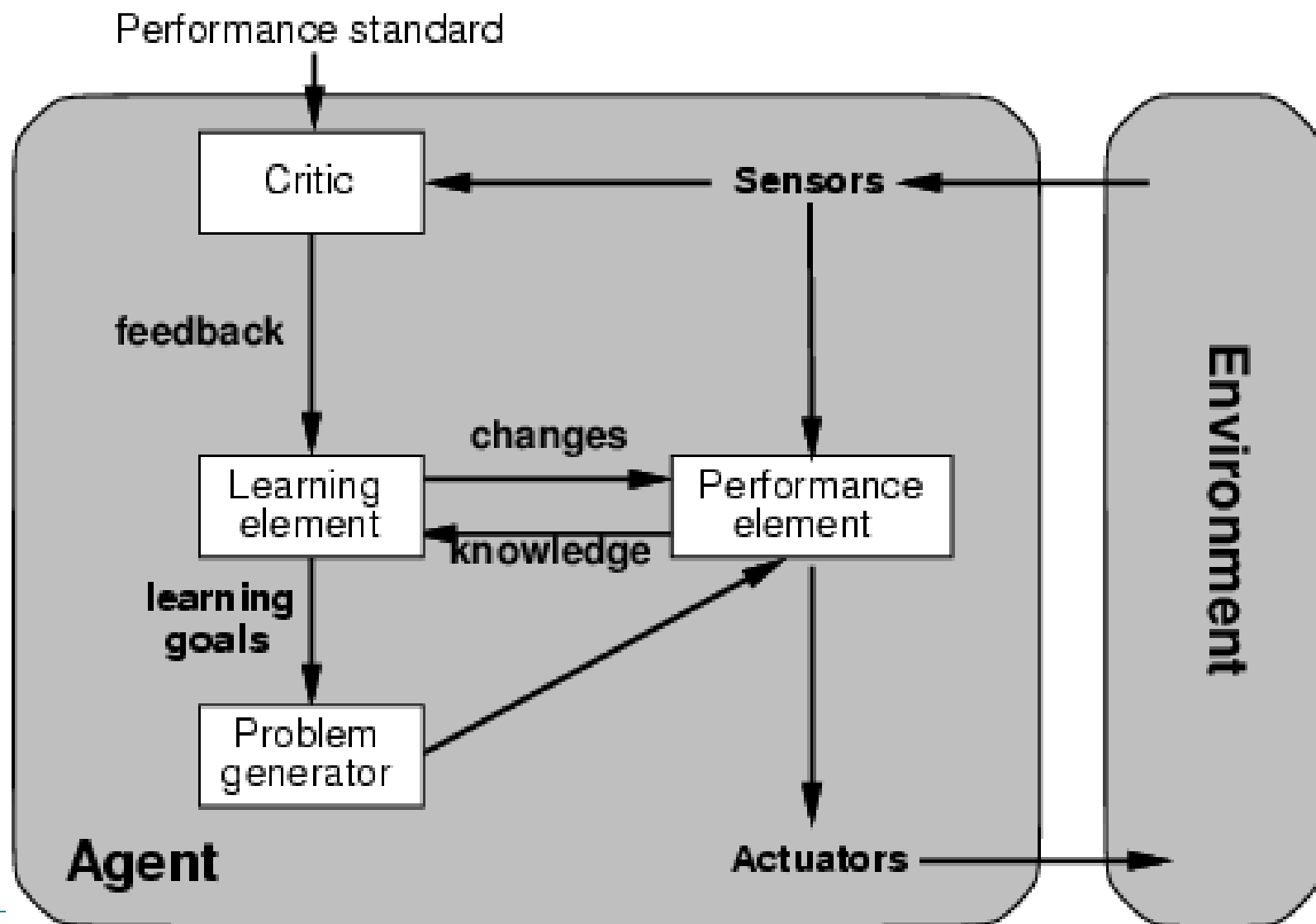
Utility-based agents



Learning agents

- Turing – instead of actually programming intelligent machines by hand, which is too much work, build learning machines and then teach them
 - Learning also allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow
-

Learning agents



Learning agents

- Learning element – responsible for making improvements
 - Performance element – responsible for selecting external actions (it is what we had defined as the entire agent before)
 - 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
 - Problem generator is responsible for suggesting actions that will lead to a new and informative experiences
-