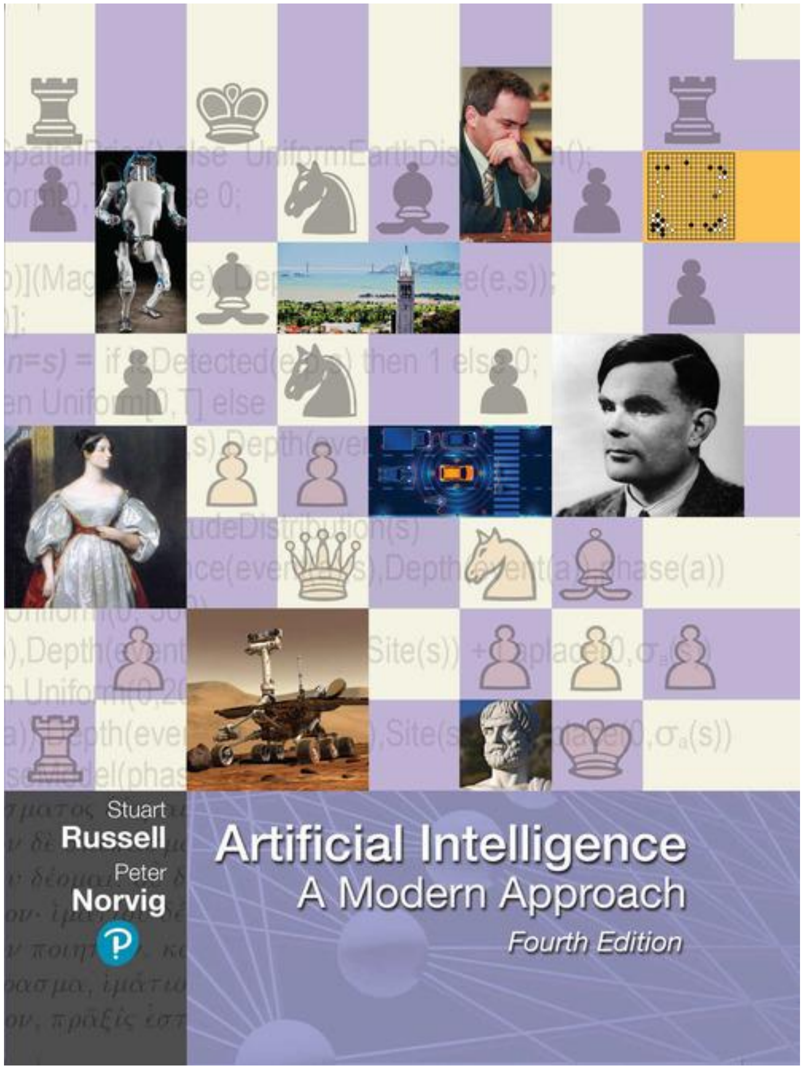
# Artificial Intelligence Fundamentals

2023-2024




"The future depends on some graduate student who is deeply suspicious of everything I have said."

— Geoffrey Hinton

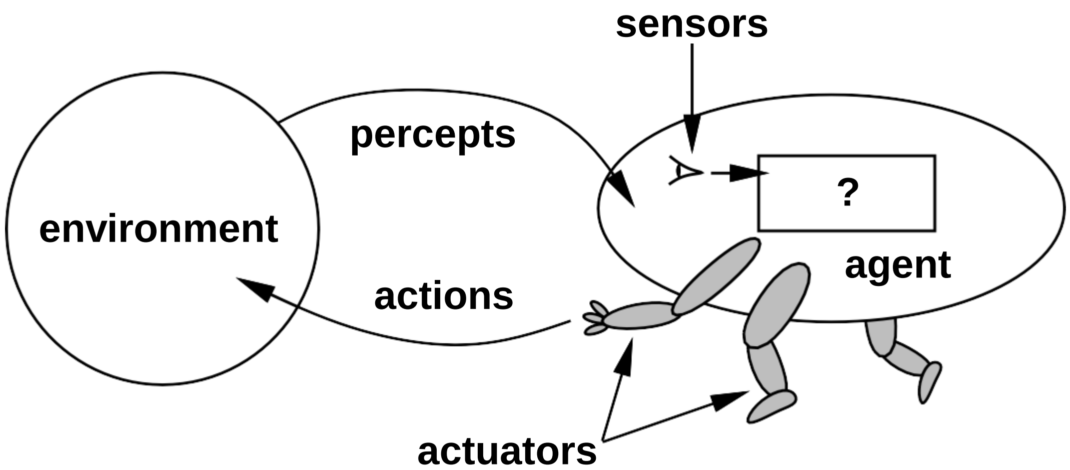
## AIMA Chapter 2

### Intelligent Agents

# Outline

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

# Agents and environments




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

An agent can be anything that can be viewed as perceiving its environment

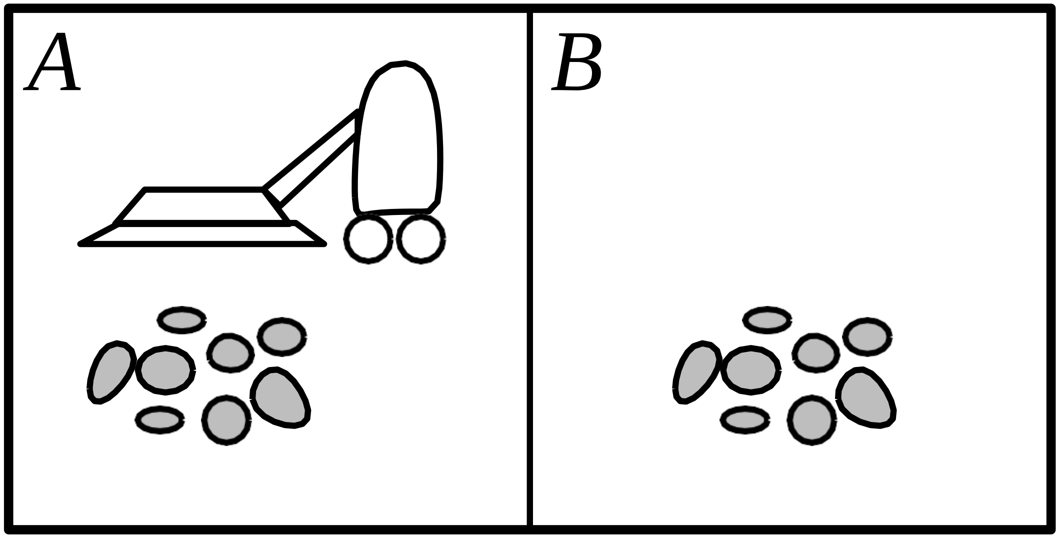
through sensors and acting upon that environment through actuators

The agent function maps from percept histories to actions:

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

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

What is the right function?
Can it be implemented in a small agent program?

| 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
```

Pearson

# Rationality

A rational agent is one that **does the right thing**.

Several different notions of the “right thing” -> AI has generally stuck to one notion called **consequentialism**: we evaluate an agent’s behavior by its **consequences**.

Recalling Norbert Wiener’s warning to ensure that *“the purpose put into the machine is the purpose which we really desire”,* notice that it can be quite hard to formulate a performance measure correctly.

Consider, for example, the vacuum-cleaner agent from the preceding section. We might propose to measure performance by the amount of dirt cleaned up in a single eight-hour shift.

A rational agent *can maximize this performance measure by cleaning up the dirt, then dumping it all on the floor, then cleaning it up again, and so on* (see “*On Superintelligence*” book by Bostrom)

# 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, we don't have access to future percepts
- Rational != successful, action outcomes may not be as expected

Rational $\Rightarrow$ exploration, learning, autonomy

# Rationality

There are **extreme cases** in which the environment is completely *known a priori* and *completely predictable*. In such cases, the agent **need not perceive or learn**; it simply **acts correctly**.

Such agents are fragile, examples:

- **Dung beetle**: After digging its nest and laying its eggs, it fetches a ball of dung to plug the entrance. If the ball of dung is removed from its grasp en route, the beetle continues its task and pantomimes plugging the nest with the nonexistent dung ball, never noticing that it is missing. ***Evolution has built an assumption into the beetle's behavior, and when it is violated, unsuccessful behavior results.***

- **Sphex wasp:** after digging a burrow, it goes out and sting a caterpillar and drag it to the burrow, enter the burrow again to check all is well, drag the caterpillar inside, and lay its eggs. The caterpillar serves as a food source when the eggs hatch. If an entomologist moves the caterpillar a few inches away while the sphex is doing the check, it will revert to the "drag the caterpillar" step of its plan and will continue the plan without modification, re-checking the burrow, even after dozens of caterpillar-moving interventions. ***The sphex is unable to learn that its innate plan is failing, and thus will not change it.***

Pearson

# The Nature of Environments: 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

# PEAS

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

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

Performance measure: safety, destination, profits, legality, comfort, . . .

Environment: US streets/freeways, traffic, pedestrians, weather, . . .

Actuators: steering, accelerator, brake, horn, speaker/display, . . .

Sensors: video, accelerometers, gauges, engine sensors, keyboard, GPS, . . .

*"the more restricted the environment, the easier the design problem"*

# Internet shopping agent

Performance measure:

Environment:

Actuators:

Sensors:

# 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)

# Environment types

| | Solitaire | Backgammon | Internet shopping | Taxi |
|---|---|---|---|---|
| Observable<br>Deterministic<br>Episodic<br>Static<br>Discrete<br>Single-agent | | | | |

# Environment types

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

# Environment types

| | Solitaire | Backgammon | Internet shopping | Taxi |
|---|---|---|---|---|
| Observable | Yes | Yes | No | No |
| Deterministic | Yes | No | Partly | No |
| Episodic | | | | |
| 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 | | | | |
| 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 | Yes | Semi | No |
| 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 | Yes | Semi | No |
| Discrete | Yes | Yes | Yes | No |
| 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 | Yes | Semi | No |
| Discrete | Yes | Yes | Yes | No |
| Single-agent | Yes | No | Yes | No |

The environment type largely determines the agent design

*The hardest case is partially observable, multiagent, nondeterministic, sequential, dynamic, continuous.*

The real world falls into this category (of course).

# Agents Structure

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 agent = architecture + program***

Most of this course is about designing agent programs.

Further reading: *[“On the Measure of Intelligence”](), F. Chollet, 2019, arXiv pre-print.*

# TABLE-DRIVEN-AGENT

**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.

| 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* |
| . | . |

It is instructive to consider why the table-driven approach to agent construction is doomed to failure. Let $\mathcal{P}$ be the set of possible percepts and let $T$ be the lifetime of the agent (the total number of percepts it will receive). The lookup table will contain $\sum_{t=1}^{T} |\mathcal{P}|^t$ entries. Consider the automated taxi: the visual input from a single camera (eight cameras is typical) comes in at the rate of roughly 70 megabytes per second (30 frames per second, $1080 \times 720$ pixels with 24 bits of color information). This gives a lookup table with over $10^{600,000,000,000}$ entries for an hour's driving. Even the lookup table for chess—a tiny, well-behaved fragment of the real world—has (it turns out) at least $10^{150}$ entries. In comparison, the number of atoms in the observable universe is less than $10^{80}$. The daunting size of these tables means that (a) no physical agent in this universe will have the space to store the table; (b) the designer would not have time to create the table; and (c) no agent could ever learn all the right table entries from its experience.

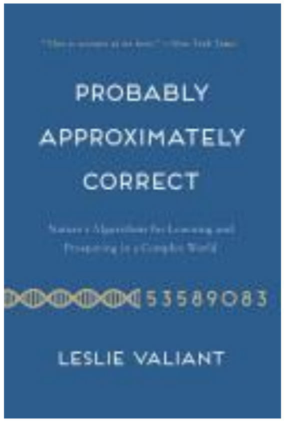
# TABLE-DRIVEN-AGENT

Despite all this, TABLE-DRIVEN-AGENT **does do** what we want, assuming the table is filled in correctly: it implements the desired agent function.

***The key challenge for AI is to find out how to write programs that, to the extent possible, produce rational behavior from a smallish program rather than from a vast table.***

This also introduce a recurring concept within our course: the ***effectiveness-efficiency trade-off***

Suggested read: [“Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World”](), Leslie Valiant (Turing award 2010), 2013.



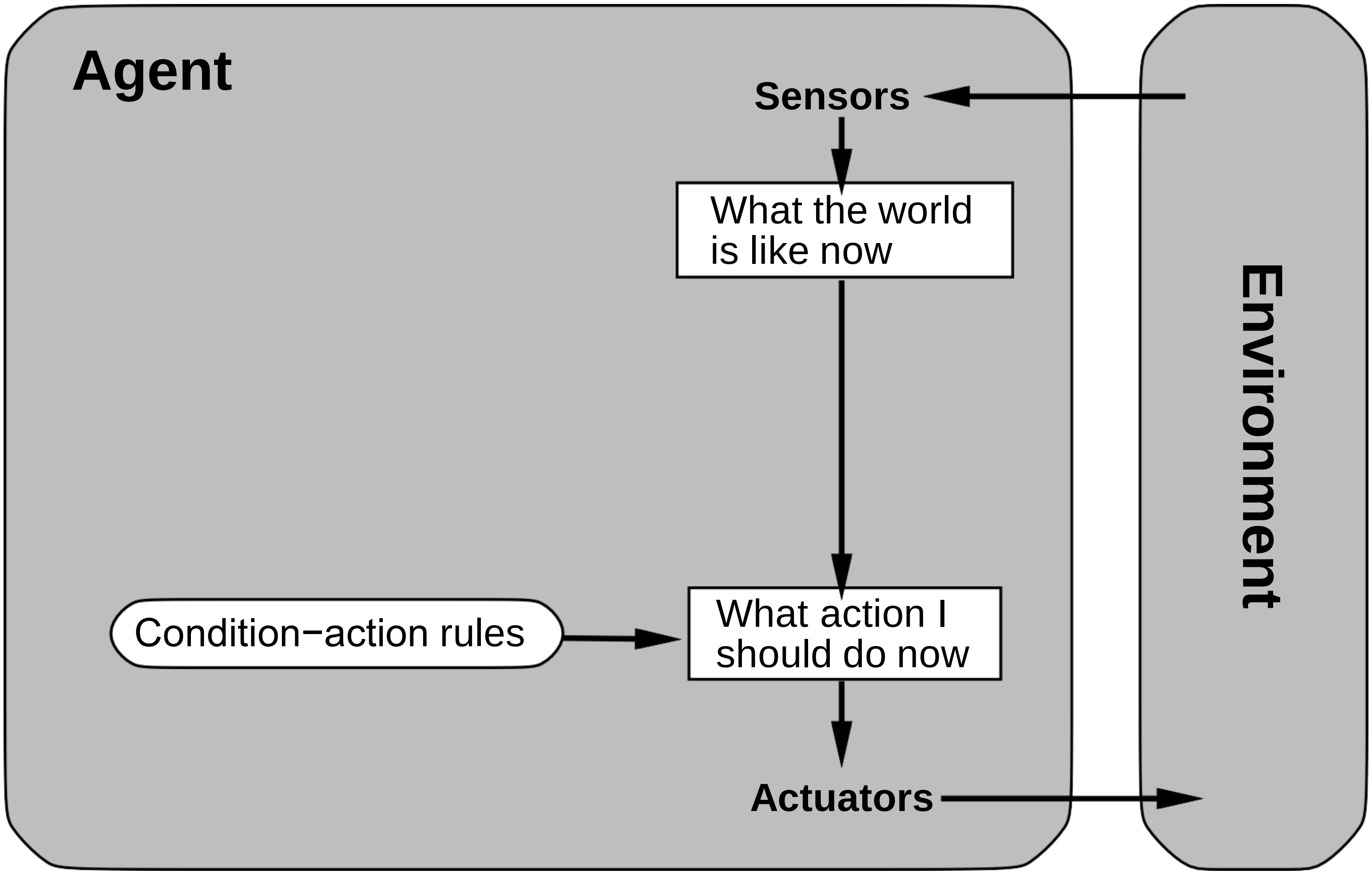
# Agent types

Four basic types in order of increasing generality:

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

*Each kind of agent program combines particular components in particular ways to generate actions*

# Simple reflex agents




These agents select **actions on the basis of the current percept**, ignoring the rest of the percept history

# 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
```

# 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*

```
loc_A = (0, 0)
loc_B = (1, 0)

"""We change the simpleReflexAgentProgram so that it does
def SimpleReflexAgentProgram():
    """This agent takes action based solely on the percep

    def program(percept):
        loc, status = percept
        return ('Suck' if status == 'Dirty'
                else'Right' if loc == loc_A
                        else'Left')
    return program


# Create a simple reflex agent the two-state environment
program = SimpleReflexAgentProgram()
simple_reflex_agent = Agent(program)
```

## AGENT

Let us now see how we define an agent. Run the next cell to see how `Agent` is defined in agents module.

```
psource(Agent)
```

The `Agent` has two methods.

- `__init__(self, program=None)` : The constructor defines various attributes of the Agent. These include
  - `alive` : which keeps track of whether the agent is alive or not
  - `bump` : which tracks if the agent collides with an edge of the environment (for eg, a wall in a park)
  - `holding` : which is a list containing the `Things` an agent is holding,
  - `performance` : which evaluates the performance metrics of the agent
  - `program` : which is the agent program and maps an agent's percepts to actions in the environment. If no implementation is provided, it defaults to asking the user to provide actions for each percept.
- `can_grab(self, thing)` : Is used when an environment contains things that an agent can grab and carry. By default, an agent can carry nothing.

# Example

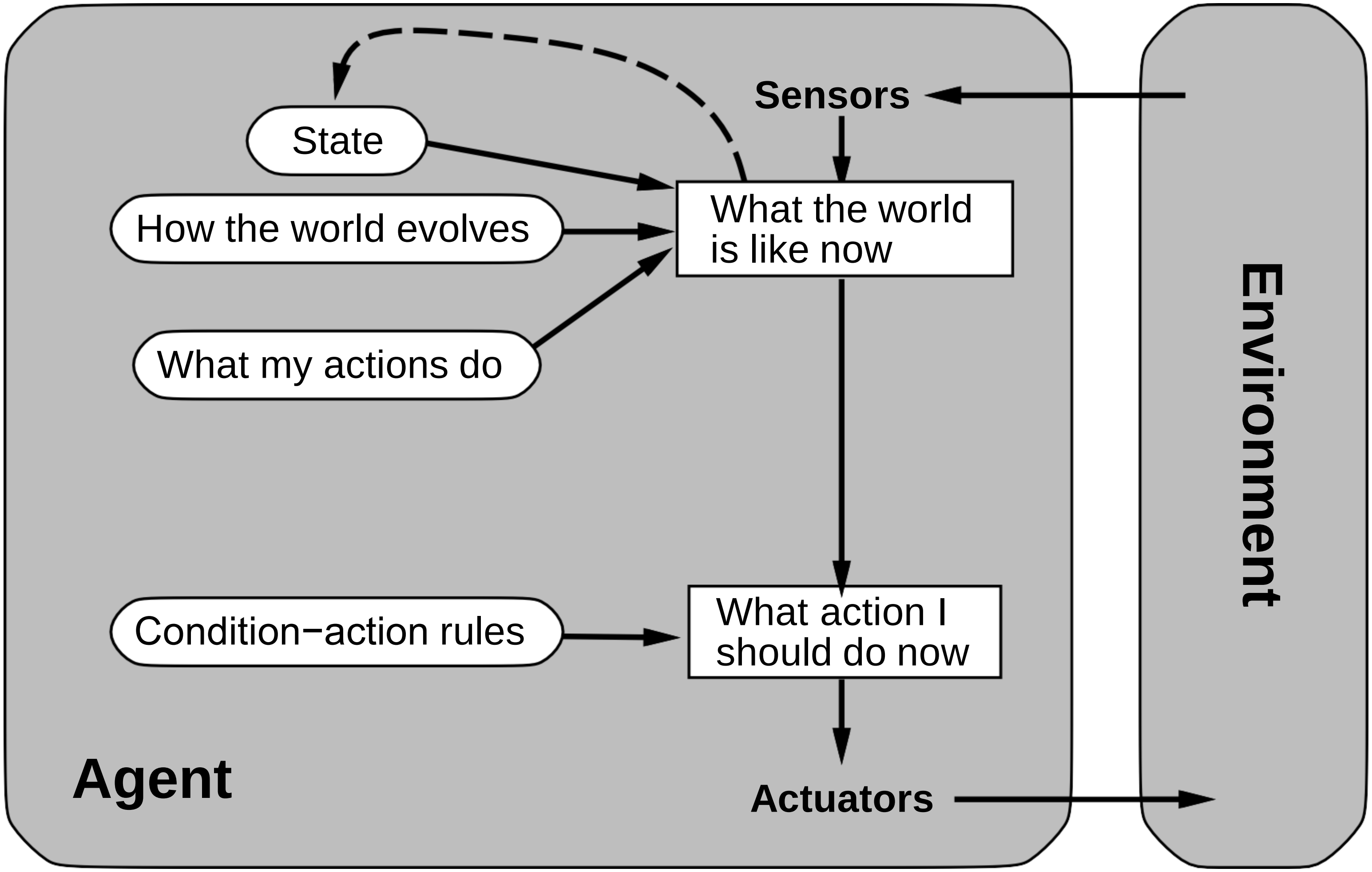
function Reflex-Vacuum-Agent( [*location*,*status*]) returns an action
  if *status* = *Dirty* then return *Suck*
  else if *location* = *A* then return *Right*
  else if *location* = *B* then return *Left*

Far better than the **table-driven-agent**:

- The most obvious reduction comes from ignoring the percept history, which cuts down the number of relevant percept sequences from **4^T** to just **4**.
- A further, small reduction comes from the fact that when the current square is dirty, the action does not depend on the location.

**Problem**: it only works if decisions can be made on just the current percept and the env. is fully observable. -> ex. vacuum agent without location sensor.

# Model-based 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».

# Example

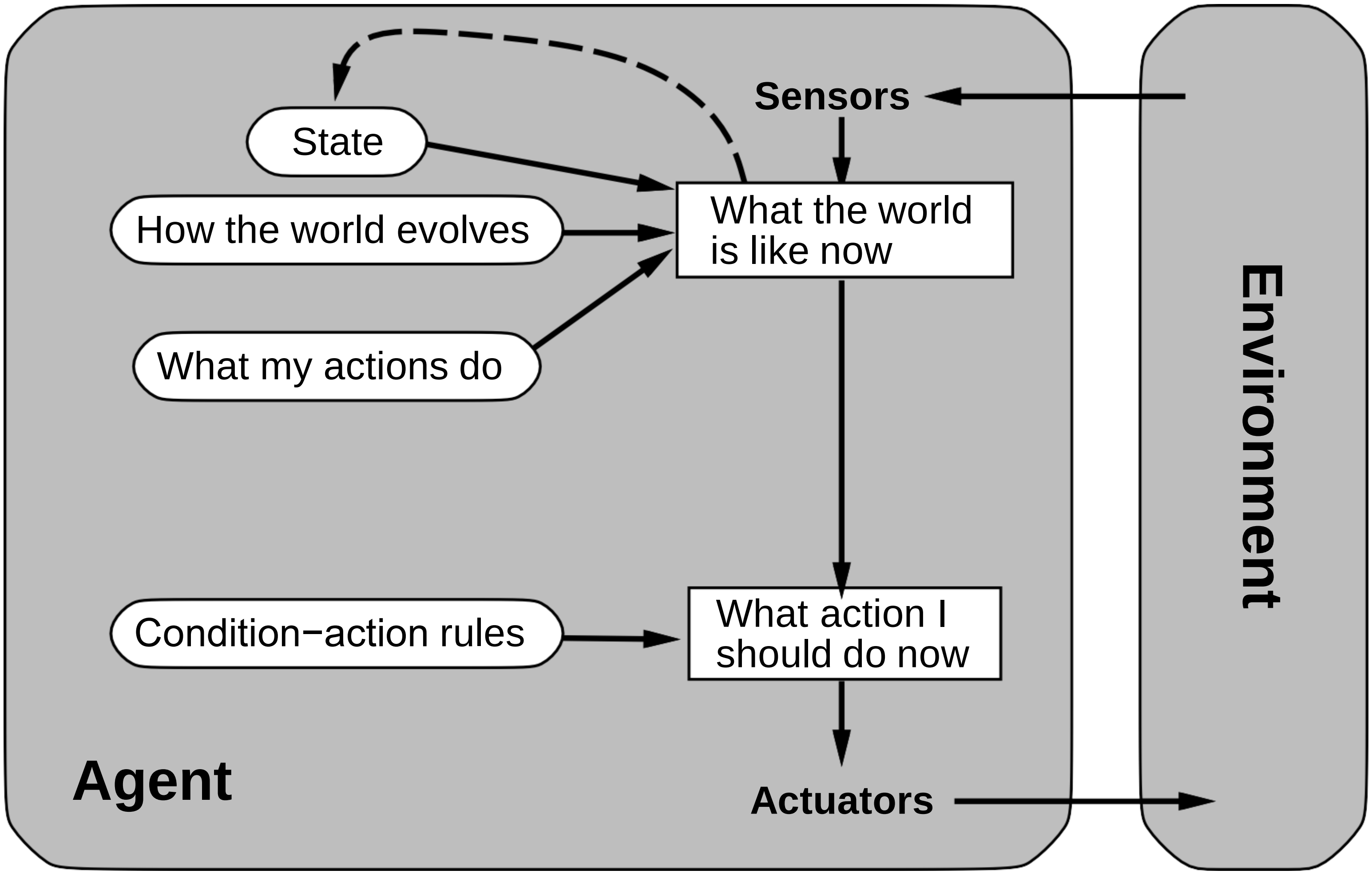
**function** MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action
  **persistent**: *state*, the agent's current conception of the world state
    *transition_model*, a description of how the next state depends on
      the current state and action
    *sensor_model*, a description of how the current 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*

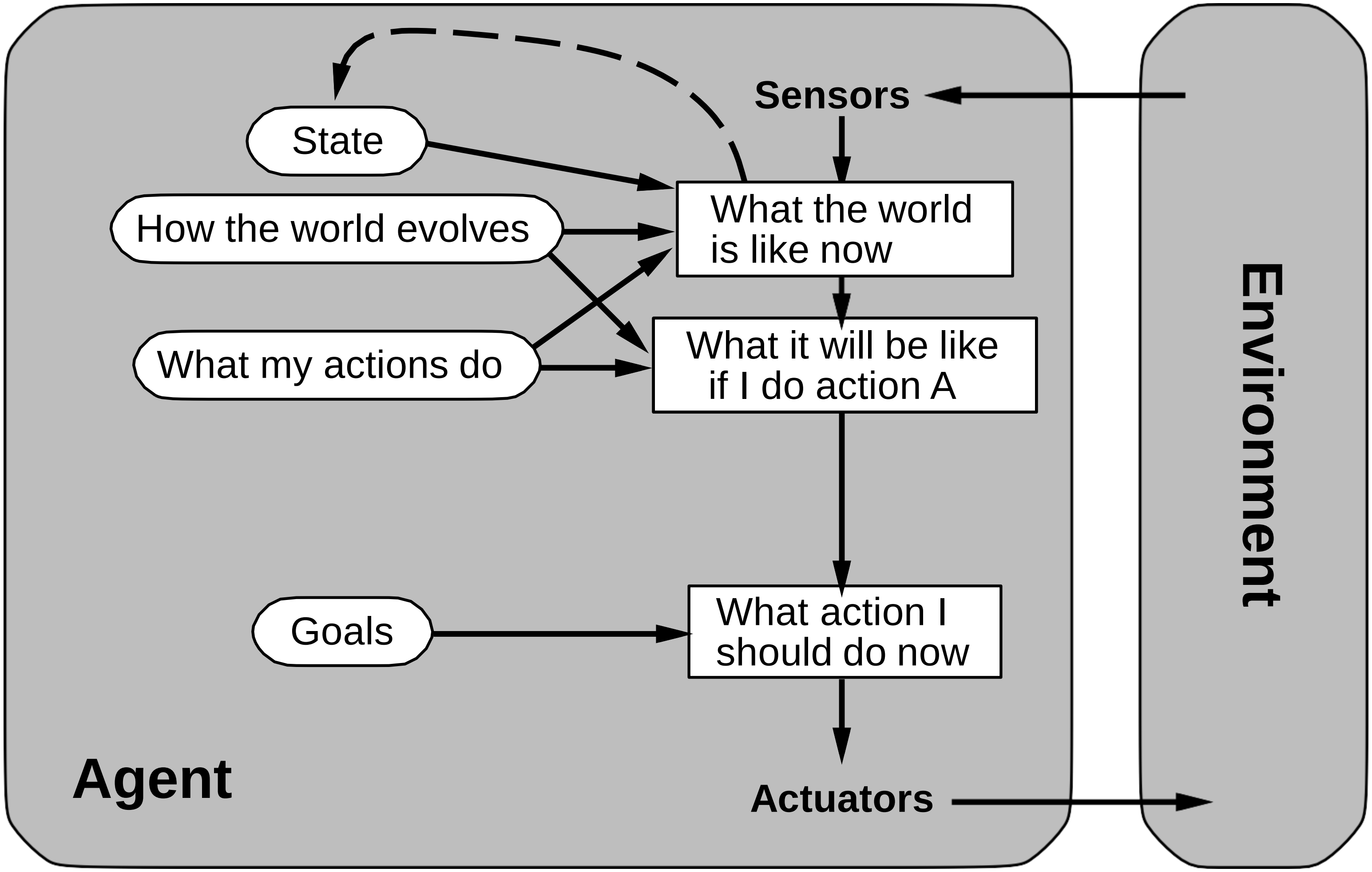
**Figure 2.12** A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

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.

# Model-based agents

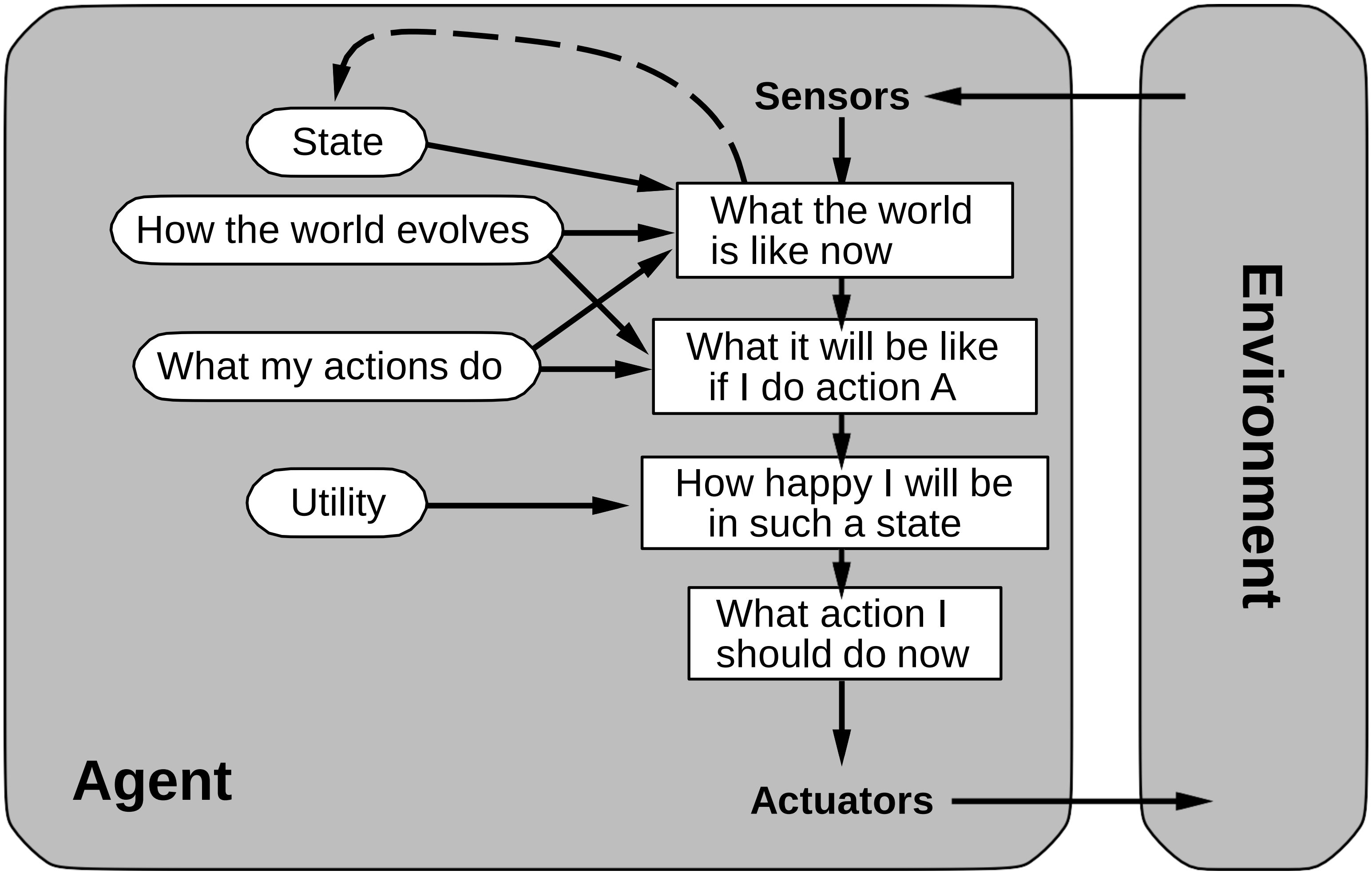


# Goal-based agents



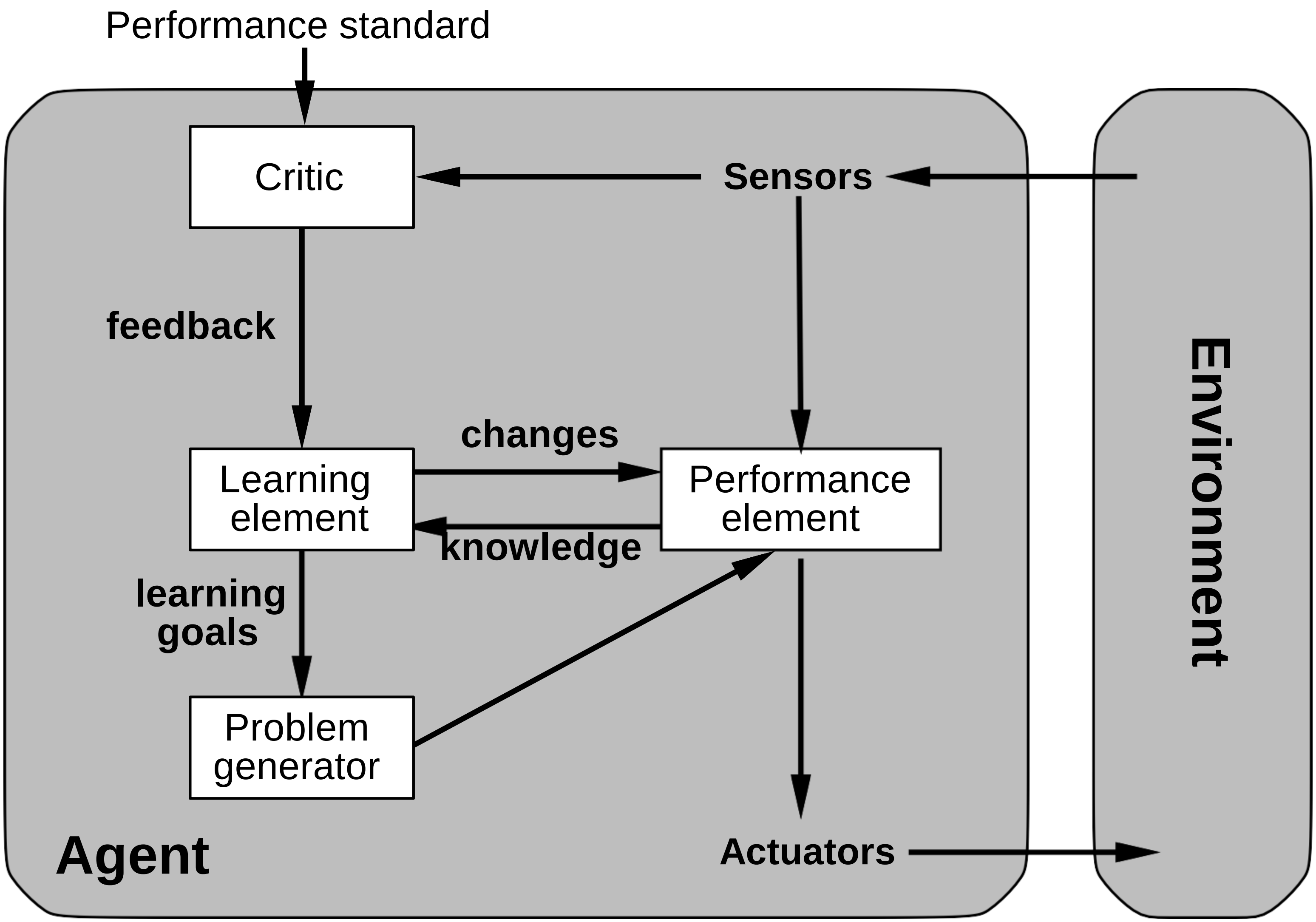

Sometimes goal-based action selection is straightforward—for example, when goal satisfaction results immediately from a single action. Sometimes it will be more tricky. **Search** and **planning** are the subfields of AI devoted to finding action sequences that achieve the agent's goals.

# Utility-based agents



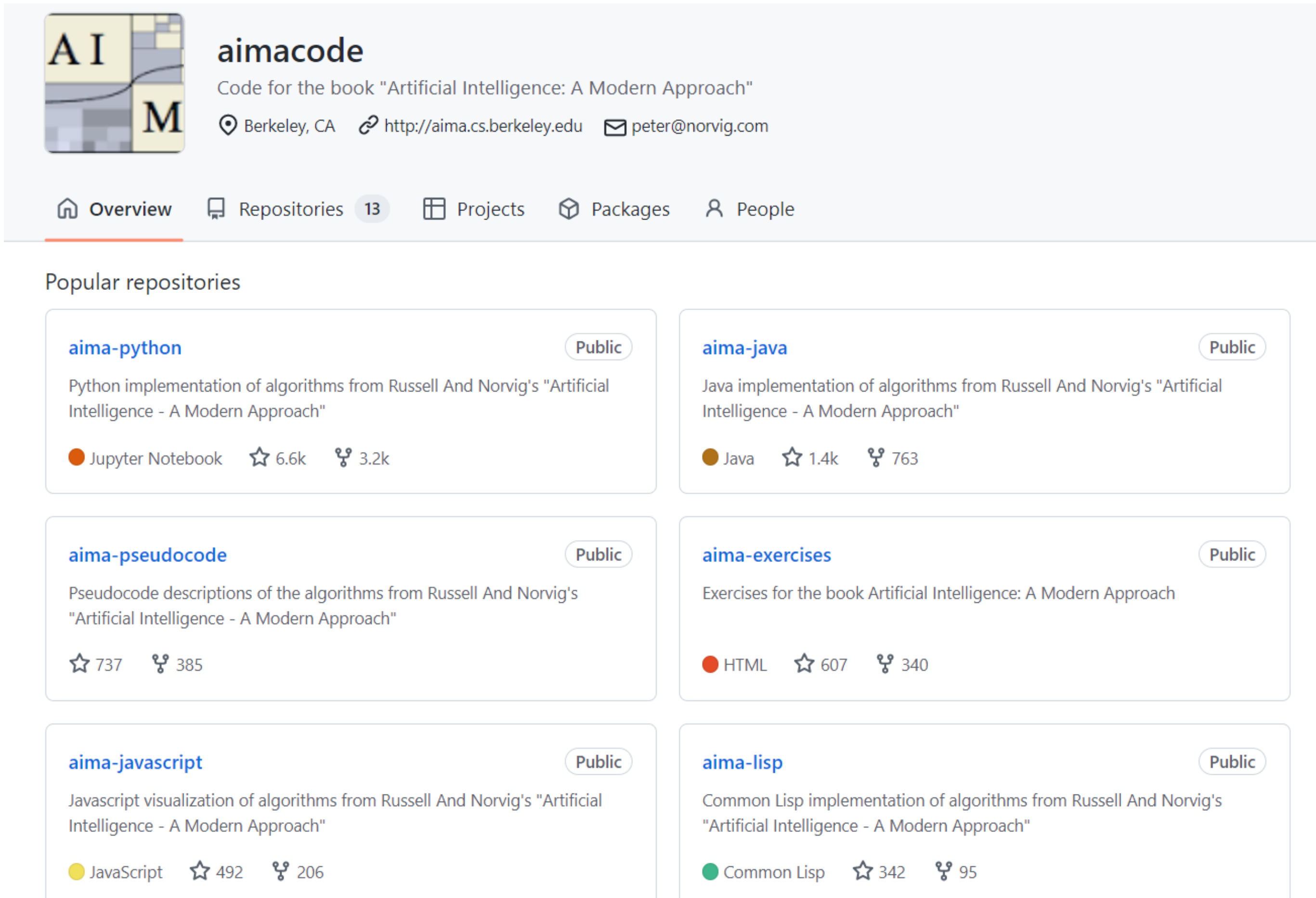

A **performance measure** assigns a score to any given sequence of environment states, so it can easily distinguish between more and less desirable ways of getting to the taxi's destination. An agent's **utility function** is essentially an internalization of the performance measure.

# Learning agents




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.

# AIMA GitHub




https://github.com/aimacode

# The Vacuum World

701 lines (701 sloc) | 29.2 KB

Raw Blame

## THE VACUUM WORLD

In this notebook, we will be discussing **the structure of agents** through an example of the **vacuum agent**. 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 **architecture**:

$$\text{agent} = \text{architecture} + \text{program}$$

Before moving on, please review **agents.ipynb**

## CONTENTS

- Agent
- Random Agent Program
- Table-Driven Agent Program
- Simple Reflex Agent Program
- Model-Based Reflex Agent Program
- Goal-Based Agent Program
- Utility-Based Agent Program
- Learning Agent

## AGENT PROGRAMS

An agent program takes the current percept as input from the sensors and returns an action to the actuators. There is a difference between an agent program and an agent function: an agent program takes the current percept as input whereas an agent function takes the entire percept history.

The agent program takes just the current percept as input because nothing more is available from the environment; if the agent's actions depend on the entire percept sequence, the agent will have to remember the percept.

We'll discuss the following agent programs here with the help of the vacuum world example:

- Random Agent Program

https://github.com/aimacode/aima-python/blob/master/vacuum_world.ipynb

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, ... *other agent architectures are possible!*

# In the next lecture...

- Projects theme & Evaluation
  - Intro to NetHack, NLE and MiniHack
  - Intro to LuckyMera
- How to create your team
- How to make a good project
  - Tools you can use
- How to write a good notebook
- How to present the project ideas / initial results