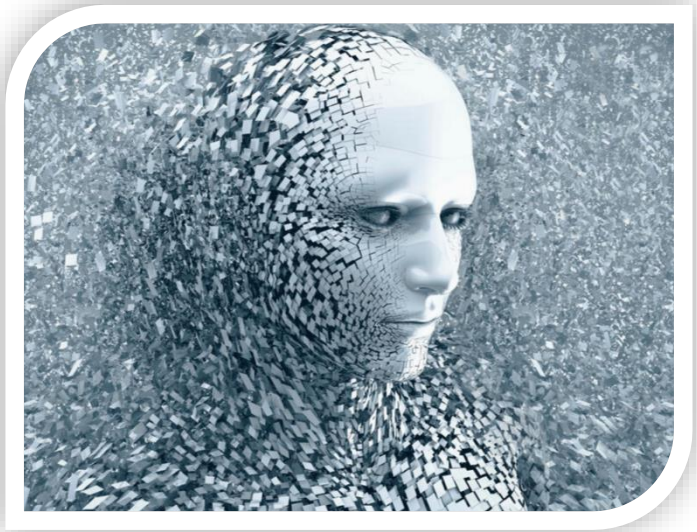
UNIVERSITY OF WEST LONDON



Artificial Intelligence

Session 2: Intelligent Agents

School of Computing and Engineering
University of West London, UK

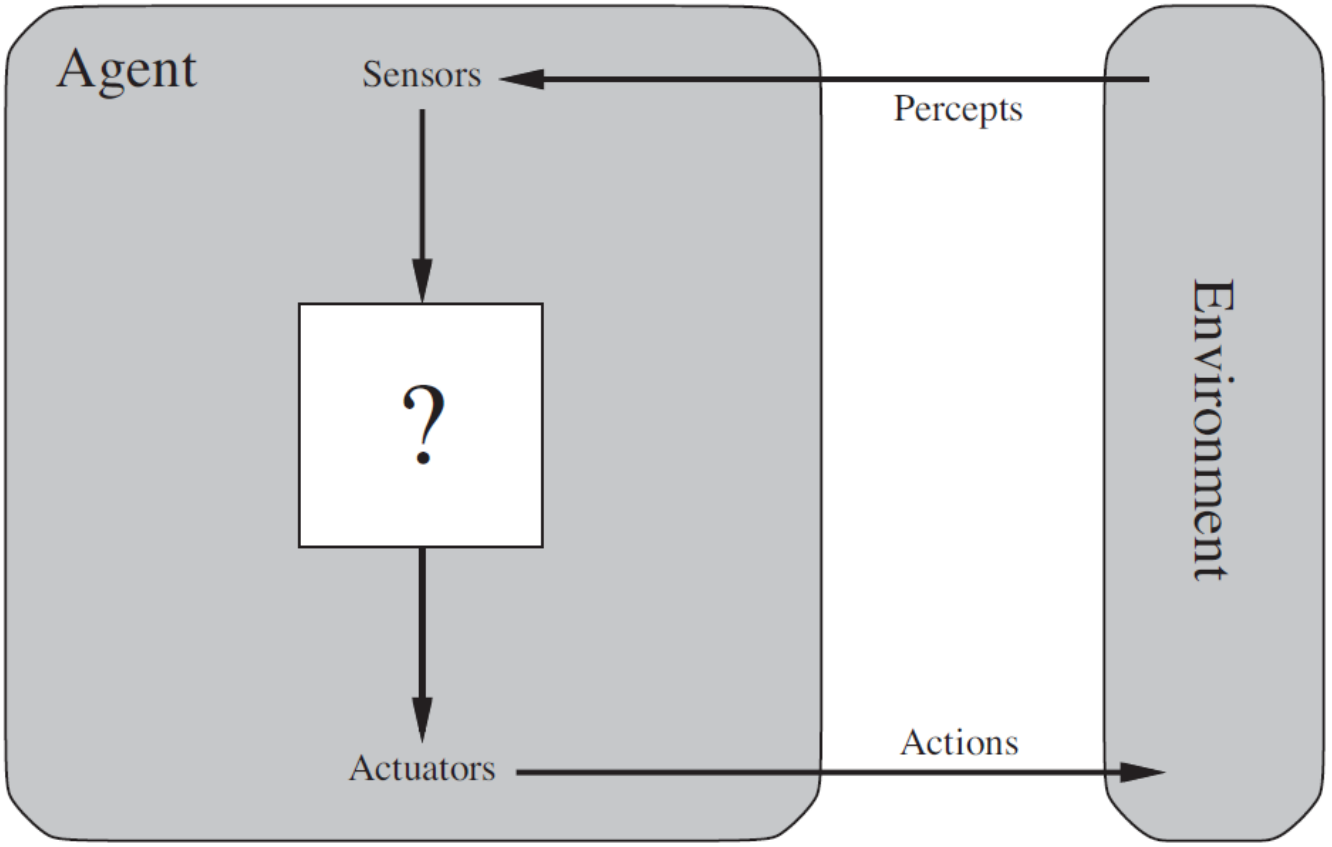
Dr Massoud Zolgharni

Overview

- Agents and environments
- Rationality: What does it mean?
- PEAS: **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors
- Types of Environments
- Types of Agents

Agent

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



Examples

Humans as agents:

- We have eyes, ears, and other organs to act as sensors
- We have hands, legs, mouth, and other body parts to use as actuators

Robotic agents:

- Robots may have: Cameras, Microphone, infrared range finders etc. as sensors
- Robots may have various tools, motors, propulsion systems etc. for actuators

Agent

Percept refers to the agent's perceptual inputs at any given instant. An agent's percept sequence is the complete history of everything the agent has ever perceived.

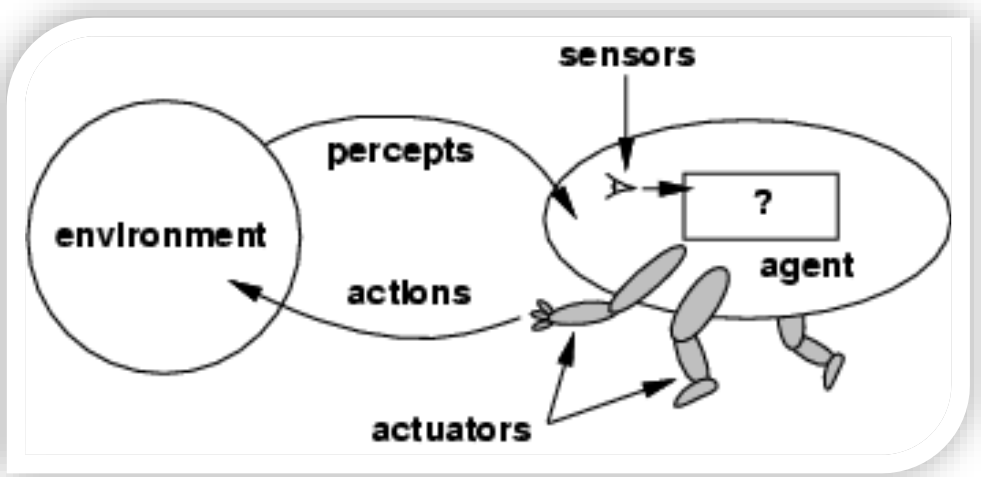
An agent's behaviour is described by the agent function that maps any given percept sequence to an action:

$$[\, f: P^* \rightarrow A \,]$$

For any given class of environments and tasks, we seek the agent (or class of agents) with the best performance

Caveat: computational limitations make perfect rationality unachievable → design best program for given machine resources

Agents and the environment



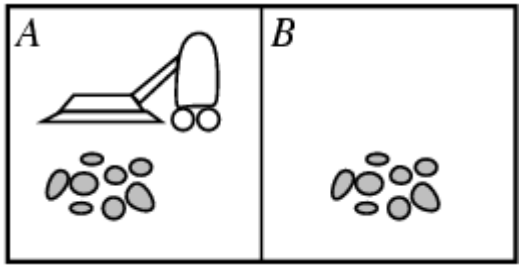
The agent function maps from percept histories to actions:

$$[\, f: P^* \rightarrow A \,]$$

The agent program runs on the physical architecture (environment) to produce *f*

A simple world (environment) & a simple agent

The Vacuum-cleaner world:



Percepts: location (A, B) and contents (Dirty, Clean)
Example for a percept: [A,Dirty]

Actions: Left, Right, Suck, NoOp

Partial tabulation of a simple agent function for the vacuum-cleaner world

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
⋮	⋮

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 the success of an agent's behaviour (actions) – *we should design performance measures according to what one actually wants in the environment*

For example: The performance measure(s) of a vacuum-cleaner agent:

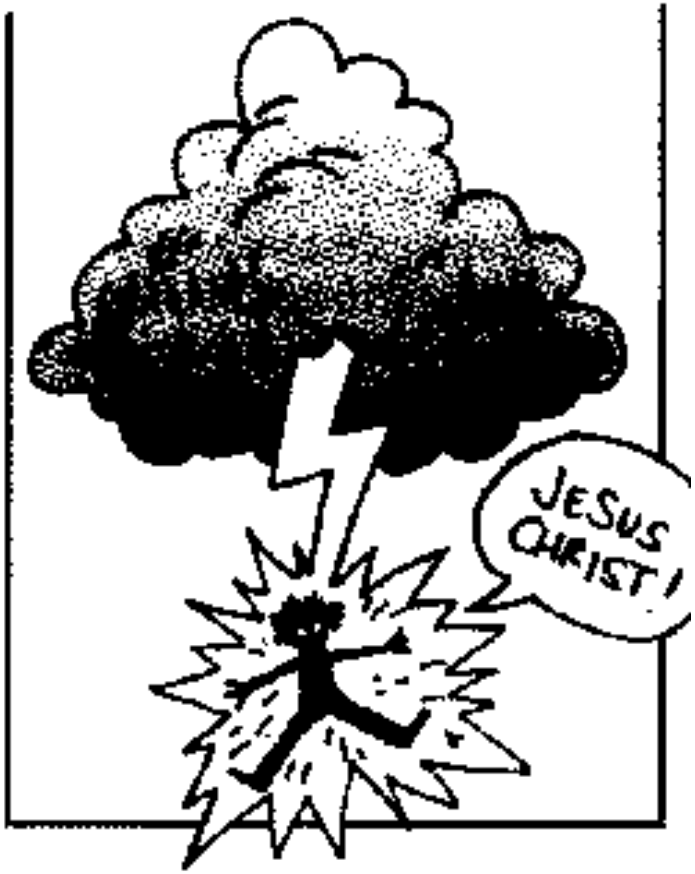
- The amount of dirt cleaned up
- The amount of time taken
- The amount of electricity consumed
- The amount of noise generated
- …

Rational Agent

For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure(s), given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Rational Agents

- Rationality is distinct from omniscience (all-knowing with infinite knowledge)
- An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality
- rationality is not the same as perfection



Rational Agents

Agents can perform actions in order to modify future percepts to obtain useful information (information gathering, exploration)

Example: exploring by vacuum-cleaning agent

An agent is autonomous if its behaviour is determined by its own experience (with the ability to learn and adapt)

e.g., a vacuum-cleaning agent that learns to foresee where and when additional dirt will appear (when agent has little or no experience, it would have to act randomly unless designer gave some assistance)

PEAS

In designing an agent, the first step must always be to specify PEAS as fully as possible

- Performance
- Environment
- Actuators
- Sensors

PEAS

Now consider an agent as a part-picking robot at an assembly line. Again, what would the PEAS for such an agent be?

- Performance: Percentage of parts in correct bins
- Environment: Conveyor belt with parts, bins
- Actuators: Jointed arm and hand
- Sensors: Camera, joint angle sensors

PEAS

PEAS: Performance measure, Environment, Actuators, Sensors

Consider, for example, the task of designing an automated car driver.
What would the following be like?

Performance measure(s) ?
Environment ?
Actuators ?
Sensors ?

Environment types 1/2

Fully observable (vs. partially observable): An agent's sensors give it access to complete state of environment at each point in time.

Deterministic (vs. stochastic): Next state of environment is completely determined by current state and action executed by agent. (If environment is deterministic except for actions of other agents, then environment is strategic)

Episodic (vs. sequential): Agent's experience is divided into atomic "episodes" (each episode consists of agent perceiving and then performing a single action), and choice of action in each episode depends only on episode itself (next episode does not depend on actions taken in previous Episodes)

e.g.,

Episodic: classification or detection agents

Sequential: automated car driver agent

Environment types 2/2

Static (vs. dynamic): Environment is unchanged while an agent is deliberating (environment is semi-dynamic if environment itself does not change with passage of time but the agent's performance score does)

e.g.,

Static: crossword puzzle

Semi-dynamic: chess played with a clock

Dynamic: automated car driver agent

Discrete (vs. continuous): A limited number of distinct, clearly defined percepts and actions.

e.g.,

Discrete: chess environment has a finite number of distinct states; Chess also has a discrete set of percepts and actions

Continuous: Taxi driving is a continuous-state; taxi-driving actions are also continuous

Single agent (vs. multi-agent): An agent operating by itself in an environment.

Heading

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

The environment type largely determines the agent design

The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent

Agent functions and programs

An agent is completely specified by the agent function mapping percept sequences to actions.

Aim: find a way to implement the rational agent function concisely.

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*
⋮	⋮

```
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
```

Table-lookup 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
```

Drawbacks of tabulating the agent function:

- Need to construct table by trying out all possible percept sequences
- Huge table, Take a long time to build the table
- No autonomy

→ produce rational behaviour from a smallish program rather than from a vast table

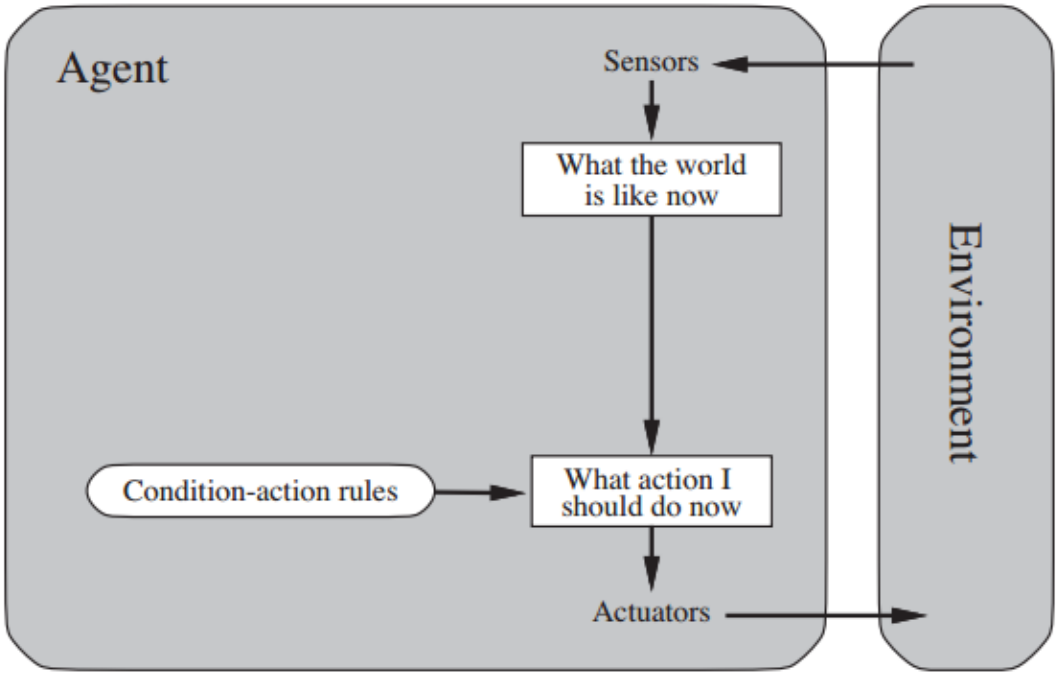
Types of Agents

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 current percept, ignoring rest of percept history



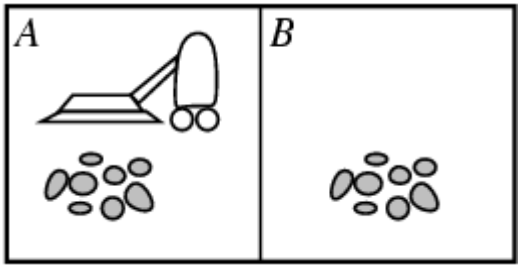
function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
  persistent: *rules*, a set of condition–action rules

  state ← INTERPRET-INPUT(*percept*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

Agent program for a vacuum-cleaner agent

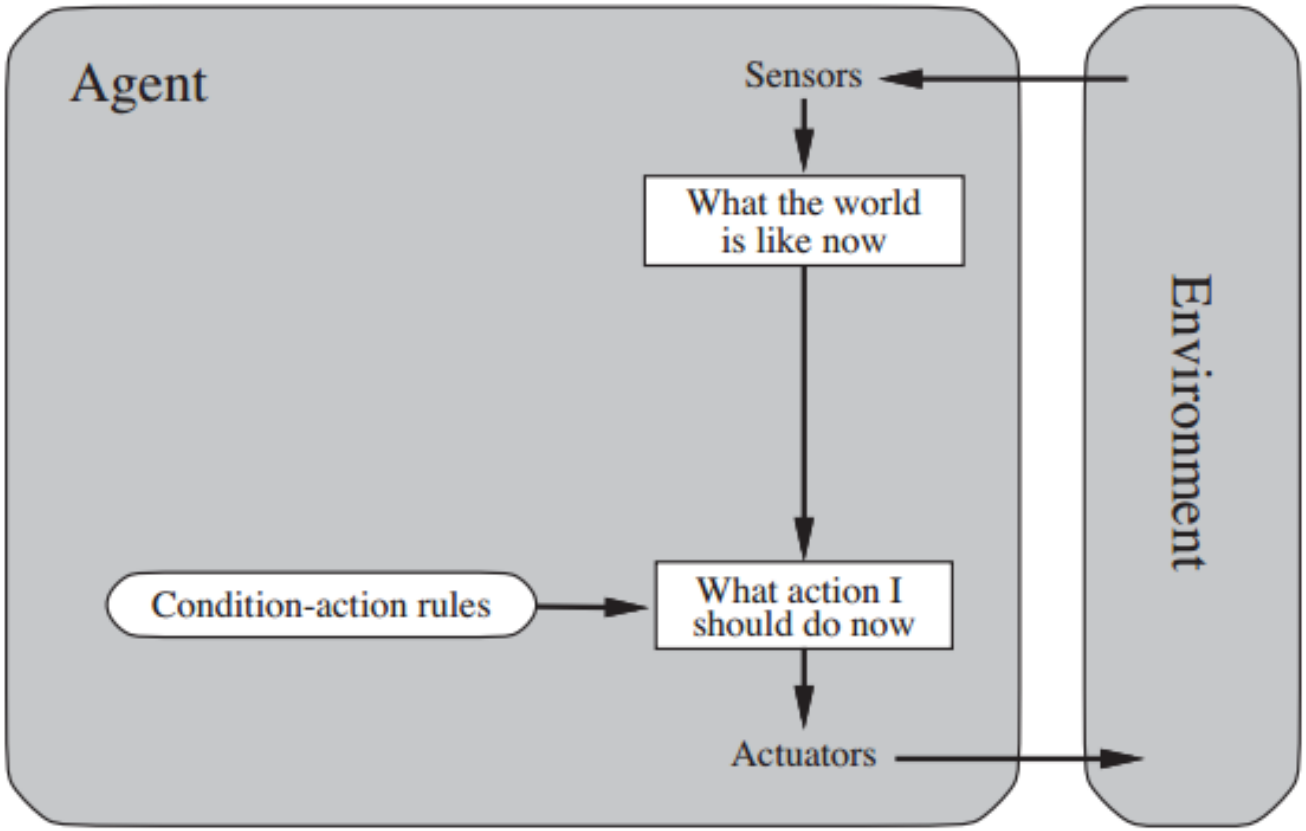
```
function REFLEX-VACUUM-AGENT([location,status]) returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

The agent program for a simple reflex agent in the two-state vacuum environment. This program implements the agent function tabulated before, remember:



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
⋮	⋮

Simple reflex agents



A simple reflex agent acts according to a rule whose condition matches the current state, as defined by the percept.

Agent program for a self-driving car agent

if *car-in-front-is-braking* **then** *initiate-braking*.

Model-based reflex agents

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:

- agent should maintain some sort of internal state that depends on the percept history
- reflects at least some of the unobserved aspects of current state

E.g., for driving tasks such as changing lanes, agent needs to keep track of where other cars are if it can't see them all at once

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: 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
  return action
```

Model-based reflex agents

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

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

Reflex agent

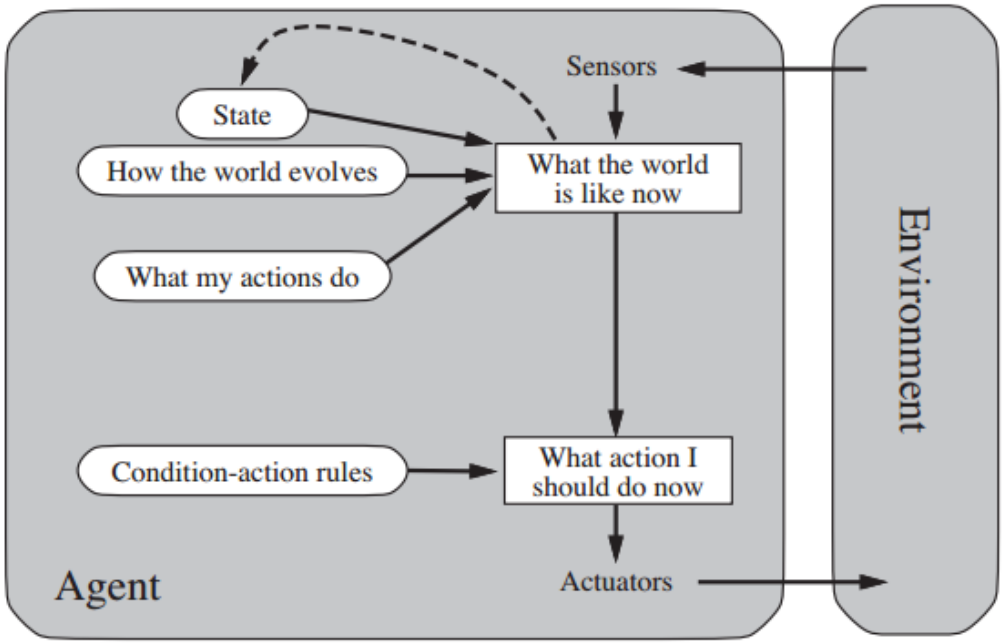
```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              model, a description of how the next state depends on current state and action
              rules, a set of condition–action rules
              action, the most recent action, initially none

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

Model-based reflex agents

Updating internal state information as time goes by requires two kinds of knowledge:

- how the world evolves independently of the agent; e.g., an overtaking car generally will be closer behind than it was a moment ago
- how agent's own actions affect the world; e.g., when agent turns steering wheel clockwise, car turns right, or that after driving for five minutes northbound on freeway, one is usually about five miles north of where one was five minutes ago



This knowledge about "how the world works" is called a **model** of the world.

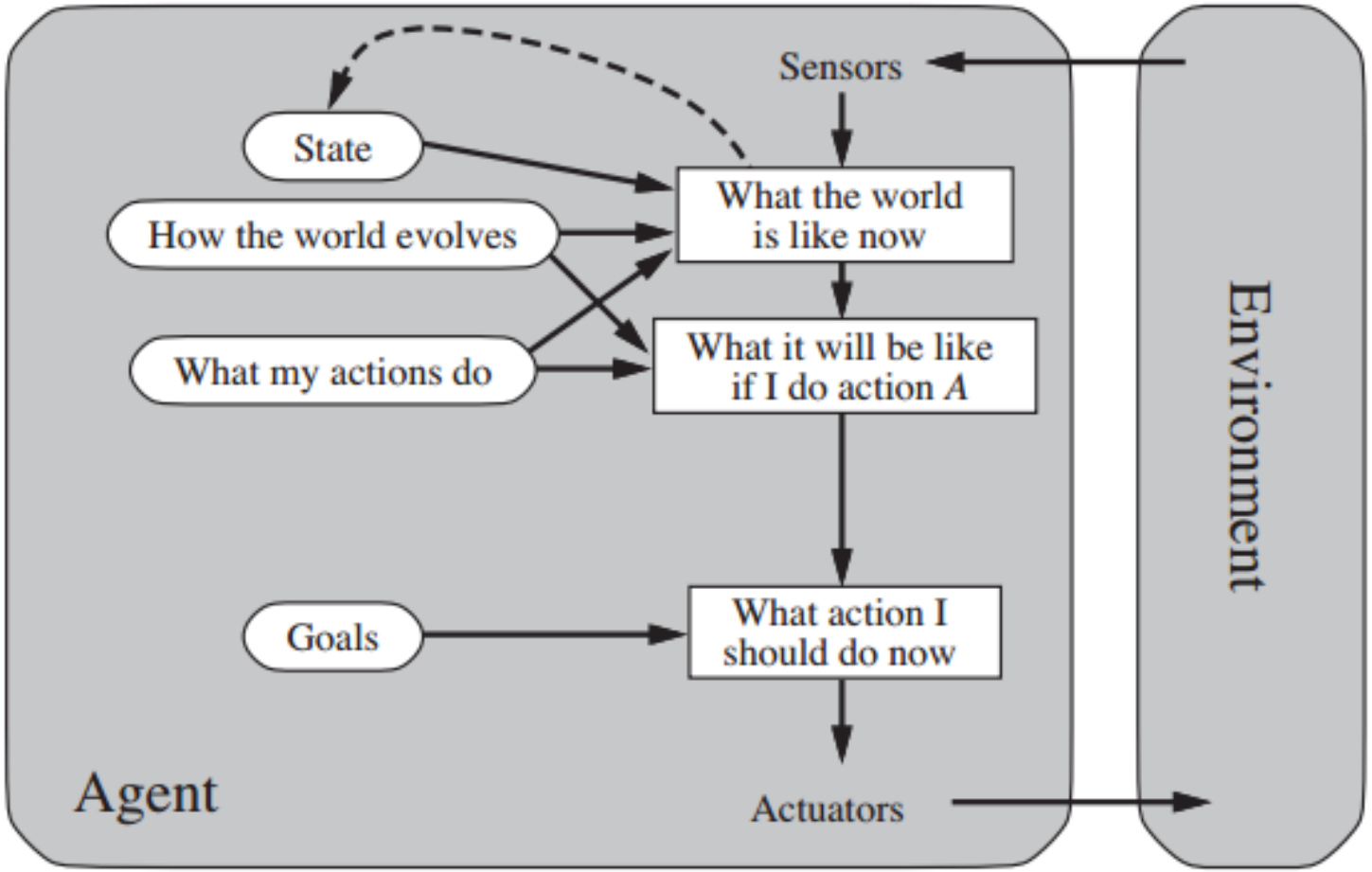
Goal-based agents

Knowing something about current state of environment is not always enough to decide what to do

e.g., at a road junction, car can turn left, right, or go straight on: correct decision depends on where car is trying to get to

As well as a current state description, agent needs some sort of goal information that describes situations that are desirable

Goal-based agents



A goal-based agent keeps track of world state as well as a set of goals it is trying to achieve, and chooses an action that will lead to achievement of its goals

Utility-based agent

Goals alone are not enough to generate high-quality behaviour in most environments.

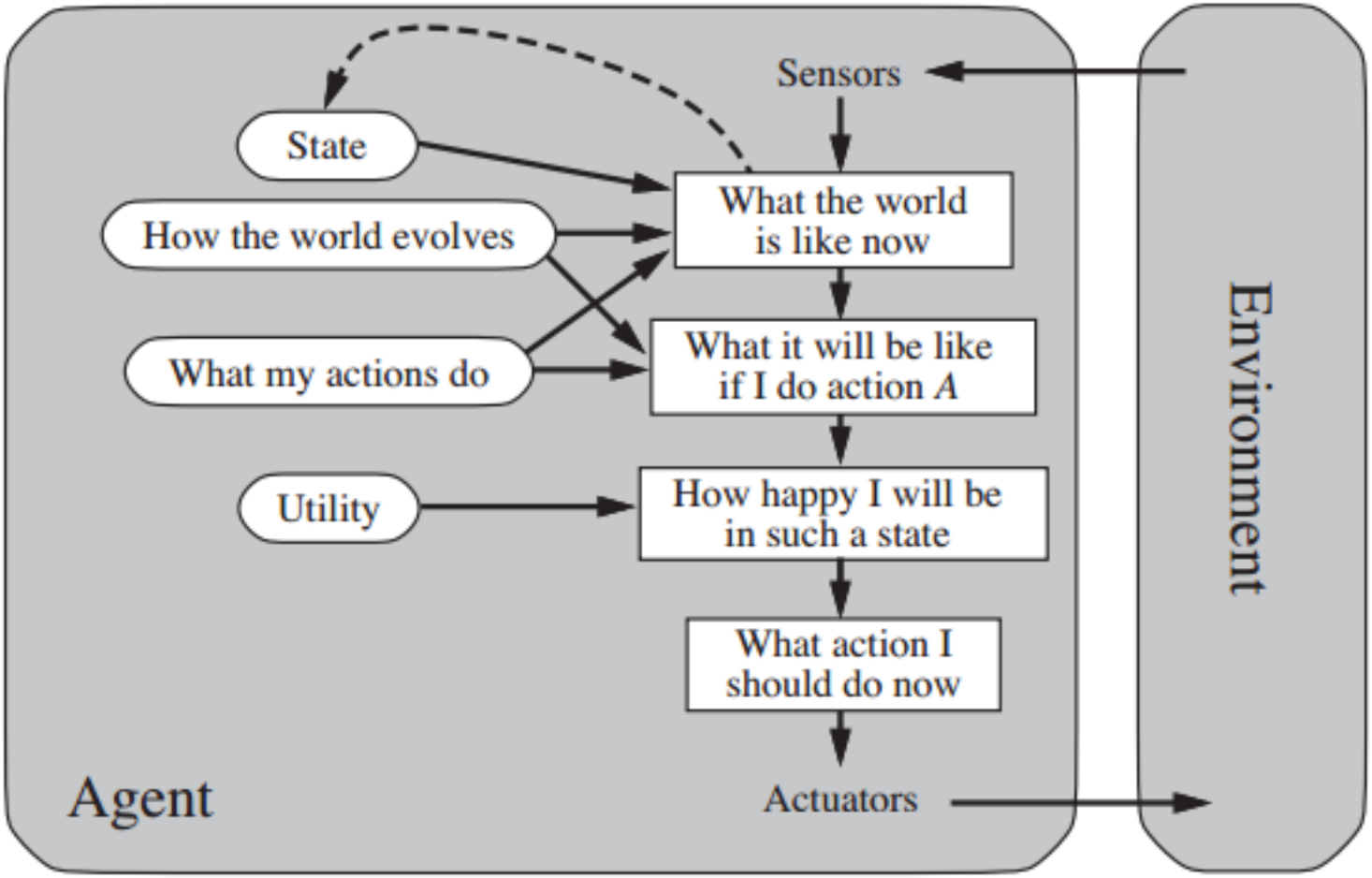
e.g., many action sequences will get the car to its destination (achieving goal) but some are quicker, safer, more reliable, or cheaper than others

Goals just provide a crude 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.

- “happy” not a scientific term → we use utility instead
- An agent’s utility function is essentially its performance measure

Utility-based agent

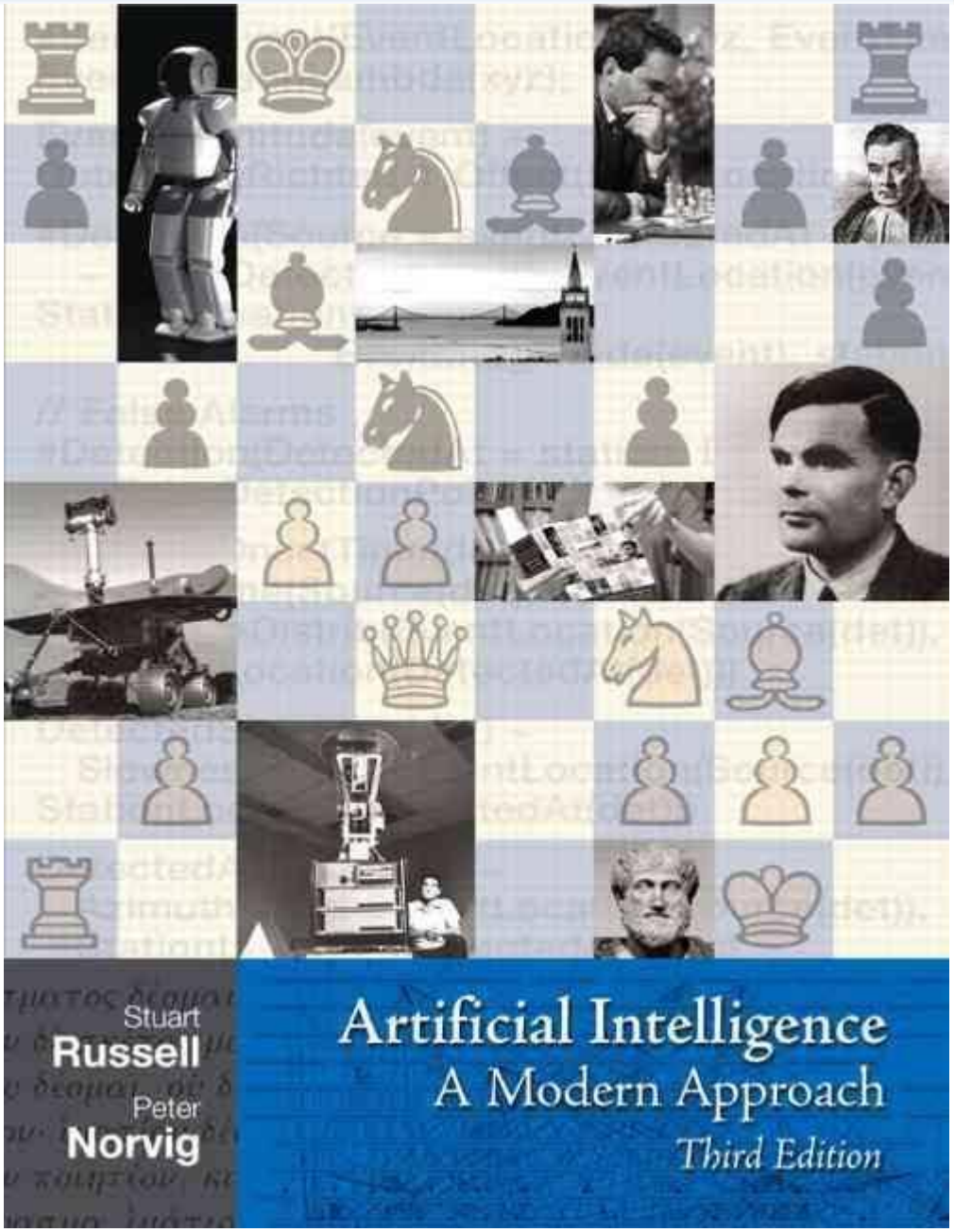


A utility-based agent uses a model of the world, along with a utility function that measures its preferences among states of the world, and chooses the action that leads to best expected utility

Recommended reading

Stuart Russell, Peter Norvig: ***Artificial Intelligence A Modern Approach***

Chapter 2



ANY Questions?