Artificial Intelligence 1

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

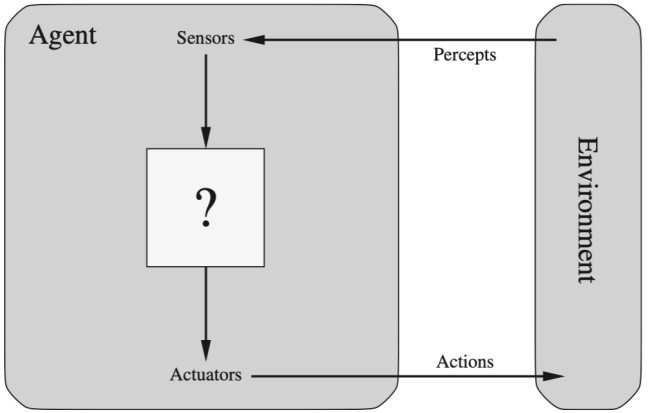
Recall, that this course will focus on **rational behaviour**:

- behaviour (not necessarily though processes),
- ideal performance (not necessarily equal to human)

We attempt to build **rational agents**

- What is an **agent**?
- What is a **rational agent**?
- How does an **agent environment** look?
- What **properties of environments** are important?
- What are skeleton **agent designs**?

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



Some examples:

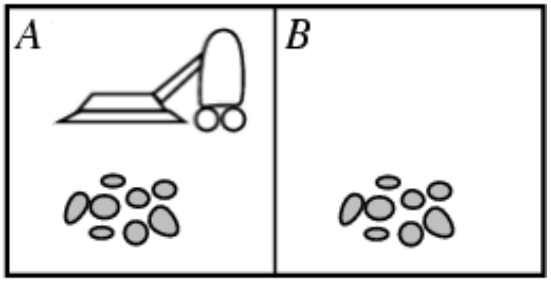
- a human agent
  - eyes, ears, nose, … → hands, legs, mouth,…
- a robotic agent
  - camera, infrared finder, … → arms, wheels, …
- a software agent
  - keyboard, network packets… → screen, sending packets,…

Agent perceives **percepts** and agent's behaviour is fully determined by the complete history of everything the agent has ever perceived.

Formally, agent's choice of action can be described by an **agent function** (table):

- $V^* \rightarrow A$, where V is a set of percepts, A is a set of actions
- The agent function can be built by observing agent's behaviour for all possible percept sequences.
  - we need "restart" capability for the agent and "enough" time and space
- The agent function is an **abstract mathematical description**.

Internally, the agent function will be implemented by an **agent program**.



Vacuum-cleaner

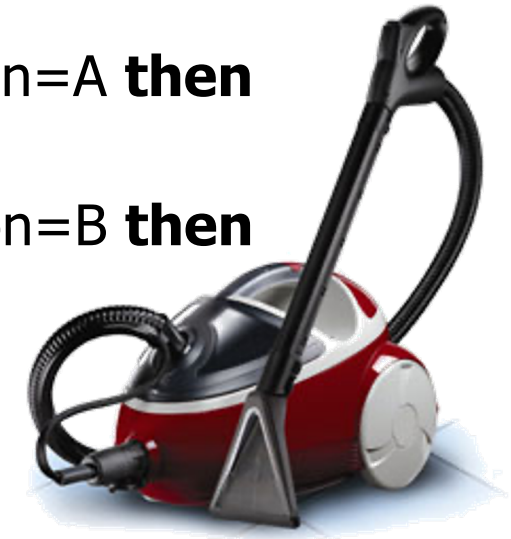
- percepts:
  location (A,B)
  property (clean, dirty)
- actions: suck up, move left, move right, do nothing

Agent function:

A sequence of percepts	**action**
(A,clean)	move right
(A,dirty)	suck up
(B,clean)	move left
(B,dirty)	suck up
(A,clean), (A,clean)	move right
…	

Agent program:

if property=dirty **then**
  suck up
else if location=A **then**
  move right
else if location=B **then**
  move left



What is the right way to fill out the table? or
What makes an agent good or bad?

We consider consequences of agent's behaviour. The notion of desirability is captured by a **performance measure** that evaluates any given sequence of environment states.

Who does usually define the performance measure?

- an agent designer

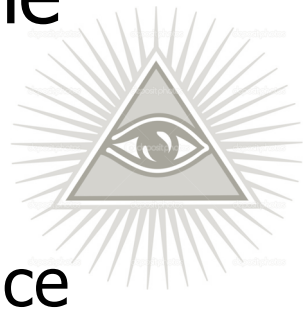
How to set the performance measure?

- 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.

Example (vacuum cleaner)

- performance measure: suck up as much dirt as possible
  - possible behaviour: suck up, dump, suck up, dump, …
- better performance measure: have a clean floor

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.



Beware, this is different from omniscience!

- rational agents maximize **expected** performance measure
- omniscient agents maximize the **actual** performance measure

A rational agent should be **autonomous** – it should learn what it can to compensate for partial or incorrect prior knowledge.

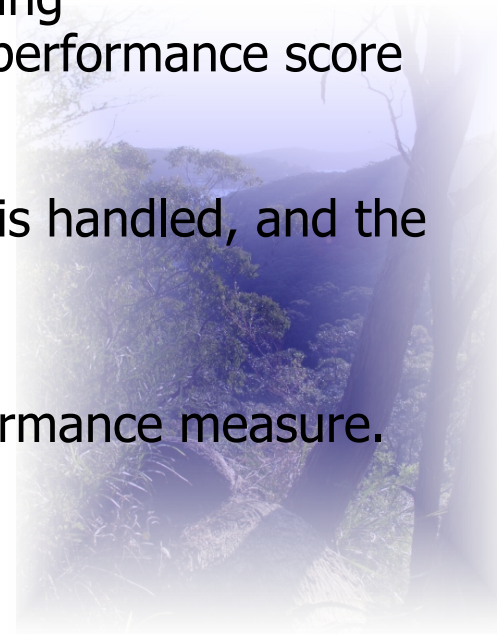
In addition to **sensors**, **actuators**, and **performance measure** agents also need **environment** to act (together this is called a **task environment**).

Example: automated taxi driver

Agent	**Perfomance measure**	**Environment**	**Actuators**	**Sensors**
Taxi driver	safe, fast, comfortable, profit	roads, other traffic, pedestrians, customer	steering accelerator, break, signal, horn	camera, sonar, lidar, odometer, GPS

Properties of task environments

- **Fully observable / partially observable**
  - agent‘s sensors give access to the complete state of environment
- **Deterministic / stochastic**
  - the next state of environment is fully determined by the current state and the action executed
  - strategic = only (other) agents can modify the environment
- **Episodic / sequential**
  - the agent’s experience is divided into atomic episodes (the next episode does not depend on actions taken in previous episodes)
- **Static / dynamic**
  - environment is not changing while an agent is deliberating
  - semidynamic = environment does not change, but the performance score does
- **Discrete / continuous**
  - depends of the state of the environment, the way time is handled, and the percepts and actions of the agent
- **Single agent / multi-agent**
  - Which entities must be viewed as agents?
    If their behaviour is best described as maximizing performance measure.
  - competitive vs. co-operative multi-agent environments



Examples of task environments

Environment	Observable	Determinististic	Episodic	Static	Discrete	Agents
Crossword puzzle	fully	deterministic	sequential	static	discrete	single
Chess with clock	fully	strategic	sequential	semi	discrete	multi
Taxi	partialy	stochastic	sequential	dynamic	continuous	multi
Image analysis	fully	deterministic	episodic	semi	continuous	single

The simplest environment

- fully observable, deterministic, episodic, static, discrete with a single agent

The most challenging environment (real-life)

- partially observable, stochastic, sequential, dynamic, continuous, multi-agent

- An **agent** is something that perceives and acts in an environment.
- The **agent function** specifies the action taken by the agent in response to any percept sequence.
- The **performance measure** evaluates the behaviour of the agent in an environment.
- A **rational agent** acts so as to maximize the expected value of the performance measure.

agent = architecture + program

- **architecture** = a computing device with physical sensors and actuators
- **program** = implementation of the agent function
  - the **mapping from percepts to actions**
  - more precisely, the agent program takes the current percept as its input (because nothing more is available from the environment) and returns an action to actuators
  - if the agent's actions depend on the entire percept sequence then the agent will have to remember the percepts
  - Obviously, the **program must be appropriate for the architecture**!

Table-driven agent

A straightforward agent that retains the complete percept sequence in memory and uses it as an index to the table with actions.

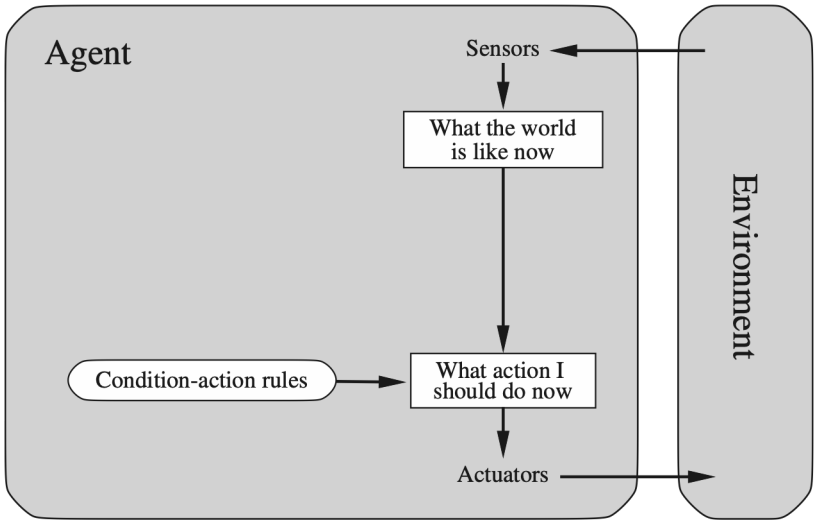
```
function TABLE-DRIVEN-AGENT(percept) returns an action
  persistent: percepts, a sequence, initially empty
              table, a table of actions, indexed by percept sequences, initially fully specified

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

Problems:

- the table is too large (even for agents working with the limited number of steps)
- the designer would not have enough time to create the table
- no agent could ever learn all the right table entries from experience
- the designer has no guidance about how to fill in the table entries

We need to find a different way!

Simple reflex agent



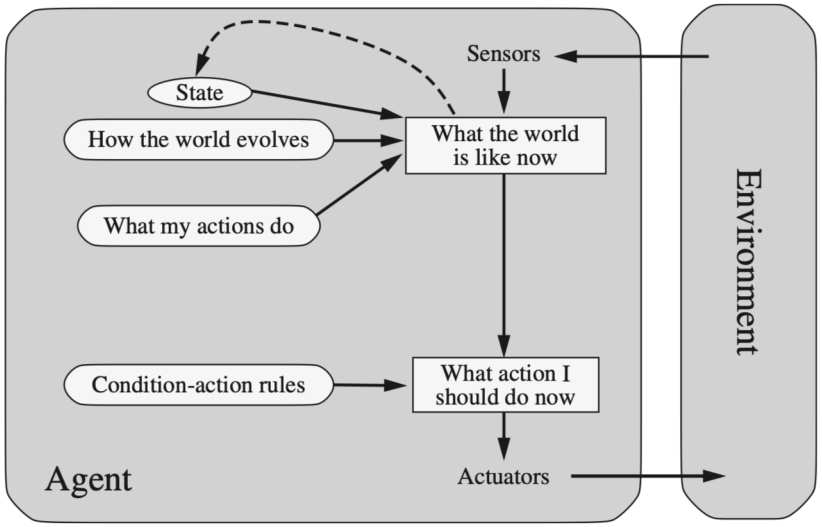
- the agent selects an action on the basis of the **current percept**
- implemented as condition-action rules (**if** property=dirty **then** suck up)
- significant reduction of the number of possibilities (to the number of percepts)

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*

- works for fully observable environments (otherwise may loop infinitely).
- randomisation of actions may help to escape from infinite loops.

Model-based reflex agent



Partial observability can be handled by keeping track of the part of world the agent cannot see now.

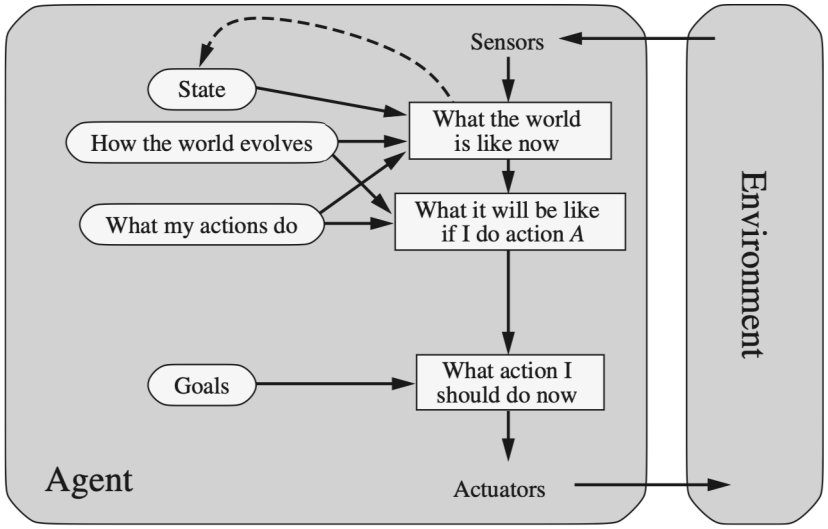
Two kind of knowledge is necessary:

- how the world evolves (independently of the agent)
- how the agent's own actions affect the world

model of the world

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

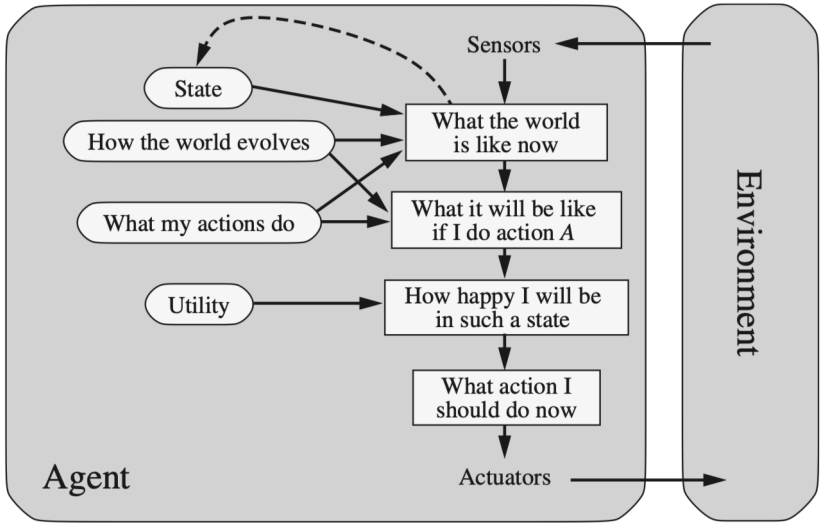
  state ← UPDATE-STATE(*state*, *action*, *percept*, *transition_model*, *sensor_model*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*



Action selection is based not only on the state but on what the agent is trying to do.

The agent needs some sort of **goal** information describing desirable situations.

- The major innovation is involving **consideration of the future**.
- **Search** and **planning** are devoted to finding action sequences that achieve the agent's goals.
- Goal-driven agent appears to be **less efficient** (than simple reflex agent), but it is **more flexible**.



Goals alone are not enough to generate high-quality behaviour in most environments (goal reached / not reached).

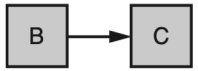
A more general performance measure allows comparison of different world states.

It is possible to map states (or their sequences) to **utility** describing the performance measure.

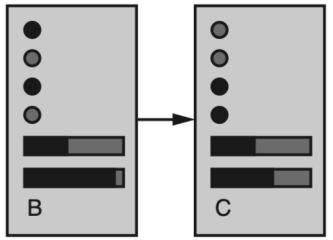
- The utility function is an **internalization of the performance measure** (the agent chooses actions to maximize its utility which will be rational if it corresponds to the external performance measure).
- The agent can perform even if there are **conflicting goals** or **chances to achieve different goals are not equal**.

How can we represent the environment that the agent inhibits?

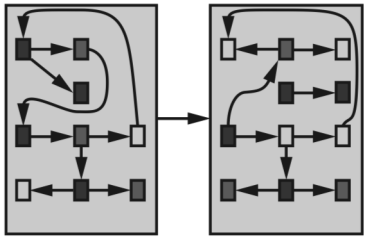
- **atomic representation**
  - each state of the world is indivisible (blackbox)
  - used in search and game-playing algorithms, Markov Decision Processes



- **factored representation**
  - each state splits into a fixed set of variables (attributes), each of which can have a value
  - used in constraint satisfaction, propositional logic, and planning



- **structured representation**
  - each state consists of a set of objects (each may have attributes) with various and varying relationships
  - used in first-order logic

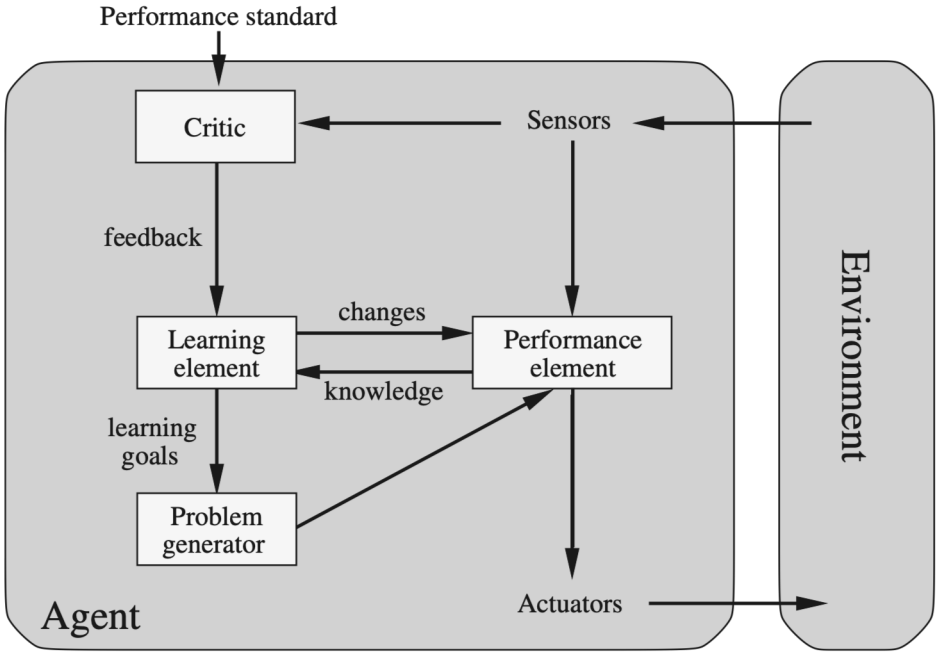


So far we described how agents select actions, but **how do we obtain the programs for action selection**?

One option is building **learning agents** and then to teach them instead of instructing them.

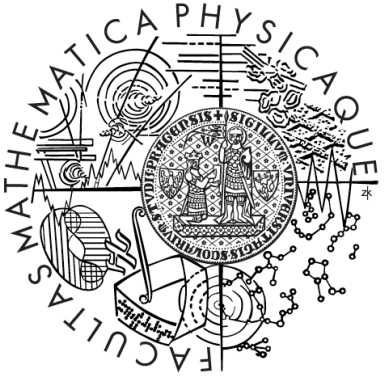
The learning agent can operate in initially unknown environment and to become more competent that its initial knowledge alone might allow.

We can extend any structure of agent to a learning agent by assuming:



- **Performance element**
  - the initial agent structure responsible for action selection
- **Learning element**
  - responsible for making improvements
- **Critic**
  - feedback on how the agent is doing (percepts themselves provide no indication of the agent's success)
- **Problem generator**
  - responsible for suggesting actions that will lead to new and informative experiences (exploratory actions)

- The **agent program** implements the agent function.
  - *Simple reflex agents* respond directly to percepts
  - *Model-based reflex agents* maintain internal state to track aspects of the world that are not evident in the current percept.
  - *Goal-based agents* act to achieve their goals.
  - *Utility-based agents* try to maximize their own expected "happiness"
- All agents can improve their performance through **learning**.
- Different components of the agent structure answer questions such as:
  - *What is the world like now?*
  - *What action should I do now?*
  - *What do my actions do?*



© 2013 Roman Barták

Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz