Technische Hochschule Rosenheim

Modul - Introduction to AI (AI1)

Bachelor Programme AAI

01 - Intelligent Agents

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

Vers. 18.10.2021

Goals (formal)

Technische Hochschule Rosenheim

- Students know the concept of rational agents as a central approach to AI.
- Students understand the wide varitey of agents operating in any imaginable enviroment.
- Students know the PEAS approach.
- Students know various basic skeleton designs.

State of the art

Technische Hochschule Rosenheim

Which of the following can be done at present?

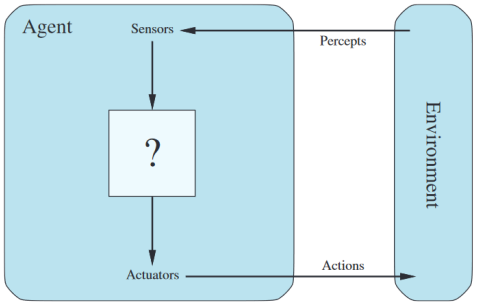
- Quadruped robots …
- Hand/arm robots …
- Driverless cars …
- Visual object classifiers …
- Answering general knowledge questions …
- Machine translation.

Agents and environments

Let's start with a general definition of AI:

> AI is the study and creation of machines that perform tasks normally associated with intelligence.

- In this course, we will use the term **agent** to refer to these machines.
- The term **agent** is very general: it covers humans, robots, softbots, thermostats, etc.
- An agent has a set of sensors, a set of actuators, and operates in an environment.



The agent function

Technische Hochschule Rosenheim

If we want, we can define an agent and its environment formally.

- We can define a set of *actions* A which the agent can perform.
- We can define a set of *percepts* P which the agent can receive.
  - Assume that there's one percept per time point.
- A simple agent function could simply map from percepts to actions:

$$f : P \rightarrow A$$

- A more complex (and general) agent function maps from percept histories to actions:

$$f : P* \rightarrow A$$

(This allows modelling of an agent with a memory for previous percepts.)

The agent program runs on the physical architecture to produce

An example

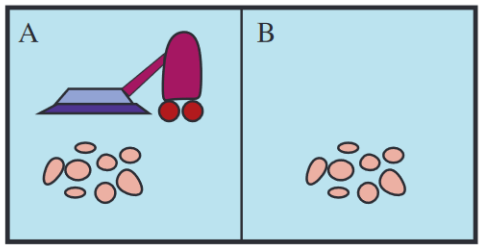
Technische Hochschule Rosenheim

The vacuum-cleaner world

Here's a simple example: a formal definition of a robot vacuum cleaner, operating in a two-room environment.

- Its sensors tell it which room it's in, and whether it's clean or dirty.
- It can do four actions: move left, move right, clean, do-nothing

How can we make it a smart agent?



Preliminaries for defining the agent function

Technische Hochschule Rosenheim

To define the agent function, we need a syntax for percepts and actions.

- Assume *percepts* have the form [location,state]: e.g. [A,Dirty].
- Assume the following four *actions*: Left , Right , Clean, NoOp.

We also need a way of specifying the function f itself.

- Assume a simple lookup table, which lists percept histories in the first column, and actions in the second column

Percept history	Action
...	...
...	...
...	...

A vacuum-cleaner agent

Technische Hochschule Rosenheim

And here's an agent program which implements this function:

```
def REFLEX-VACUUM-AGENT(location,status):
  if status = Dirty then return Clean
  else if location = A then return Right
  else if location = B then return Left
```

Evaluating the agent function

Technische Hochschule Rosenheim

It is useful to evaluate the agent function, to see how well it performs.

To do this, we need to specify a **performance measure**, which is defined as a function of the agent's environment over time.

Some example performance measures:

Evaluating the agent function

Technische Hochschule Rosenheim

It is useful to evaluate the agent function, to see how well it performs.

To do this, we need to specify a **performance measure**, which is defined as a function of the agent's environment over time.

Some example performance measures:

– 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?

Formalising the agent's environment

Technische Hochschule Rosenheim

As well as a formal description of the agent, we can give a formal description of its environment.

Environments vary along several different dimensions:

- Fully observable vs partially observable
- Deterministic vs stochastic
- Episodic vs sequential
- Offline vs online
- Discrete vs continuous
- Single-agent vs multi-agent

The environment type largely determines the agent design

Exercise 1:

Technische Hochschule Rosenheim

Can you fill? - Break out in groups and discuss.

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle Chess with a clock	1					
Poker Backgammon	2					
Taxi driving Medical diagnosis	3					
Image analysis Part-picking robot	4					
Refinery controller English tutor	Partially Partially	Single Multi	Stochastic Stochastic	Sequential Sequential	Dynamic Dynamic	Continuous Discrete

taken from http://aima.cs.berkeley.edu/figures.pdf

Environments

Technische Hochschule Rosenheim

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

taken from http://aima.cs.berkeley.edu/figures.pdf

PEAS

Technische Hochschule Rosenheim

In order to specify a scenario in which an agent performs a certain task, we need to define:

- a **P**erformance measure;
- an **E**nvironment;
- a set of **A**ctuators;
- a set of **S**ensors;

This is called the PEAS description of the agent/task.

Once the PEAS description has been given, we're ready to define the Agent function f.

Exercise 2:

Technische Hochschule Rosenheim

Can you fill? - Break out in groups and discuss.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments	Touchscreen/voice entry of symptoms and findings
Satellite image analysis system	1			
Part-picking robot	2			
Refinery controller	3			
Interactive English tutor	4			

taken from http://aima.cs.berkeley.edu/figures.pdf

PEAS Example

Technische Hochschule Rosenheim

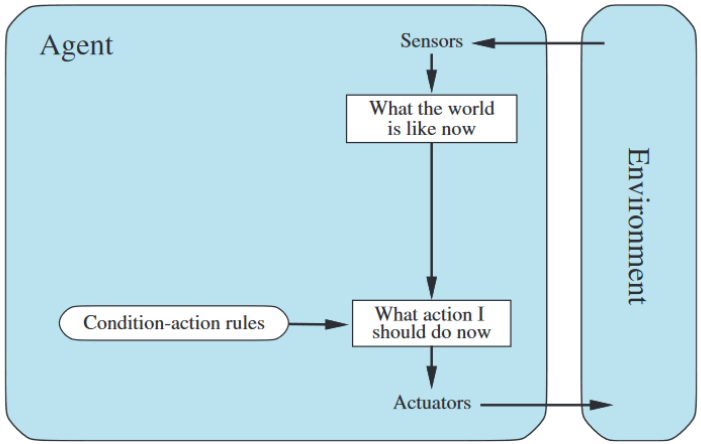
Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments	Touchscreen/voice entry of symptoms and findings
Satellite image analysis system	Correct categorization of objects, terrain	Orbiting satellite, downlink, weather	Display of scene categorization	High-resolution digital camera
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, tactile and joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, raw materials, operators	Valves, pumps, heaters, stirrers, displays	Temperature, pressure, flow, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, feedback, speech	Keyboard entry, voice

taken from http://aima.cs.berkeley.edu/figures.pdf

Simple Reflex Agent

The simplest kind of agent is the **simple reflex agent**. This agent selects actions on the basis of the current percept.

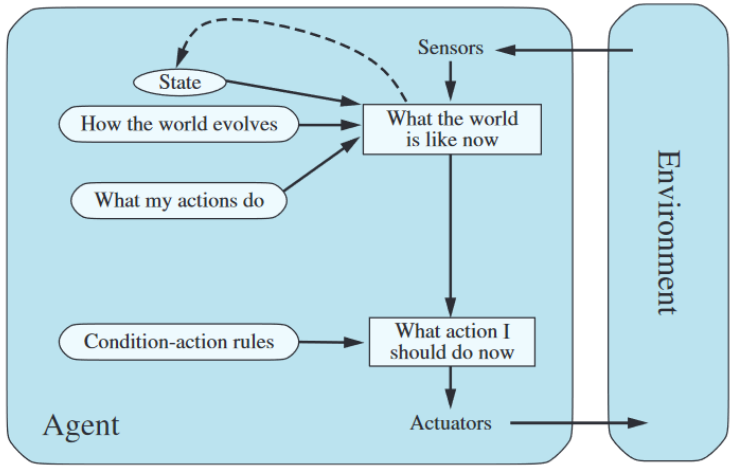
```
def SIMPLE-REFLEX-AGENT(pecept):
  ## set of condition-action rules
  rules = [{},{}...]

  state = interpretInput(percept)
  rule= ruleMatch(state, rules)
  action = rules.action
  return action
```



Model-based Reflex Agent

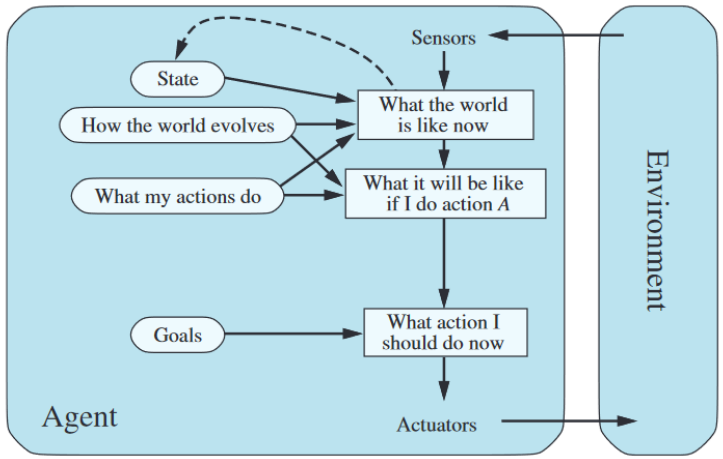
The most effective way to handle partial observability is for the agent to *keep track of the part of the world it can't see*. Therefore

```
def MODEL-BASED-REFLEX-AGENT(percept):
  ## a description of how the
  ## next state depends on
  ## the current state and action
  transition_model = []
  ## a description of how the
  ## current world state is reflected
  ## in the agent's percepts
  sensor_model = []
  ## a set of condition-action rules
  rules  = []
  ## the most recent action, initially none
  action = none
  ## the agent conception of the world
  state = updateState(state,action,percept,
    transition_model,sensor_model)
  rule = ruleMatch(state,rules)
  action = rule.action
  return action
```



Goal-based Agent

Knowing about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, right, or go straight on. The right decision depends on where the taxi is trying to get to. In other words, as well as a current state description, the agent needs some sort of **goal information**, which describes situations that are desirable — for example, being at the passenger's destination.

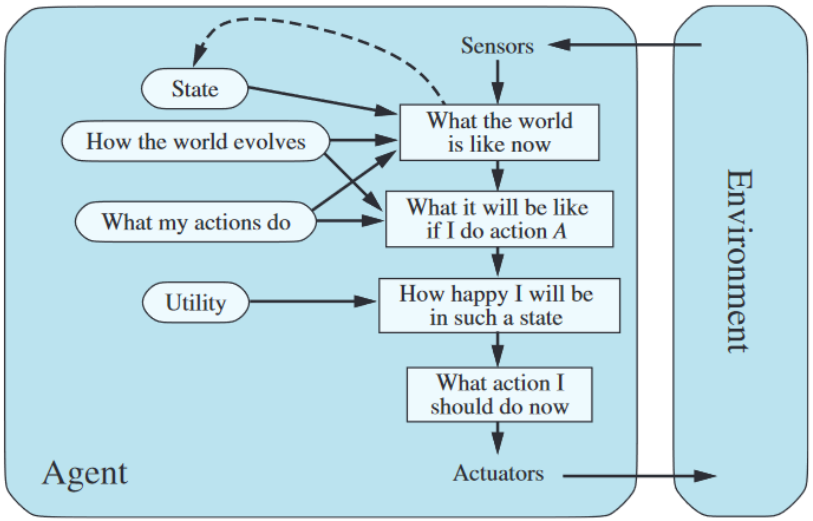


Utility-based Agent

Technische Hochschule Rosenheim

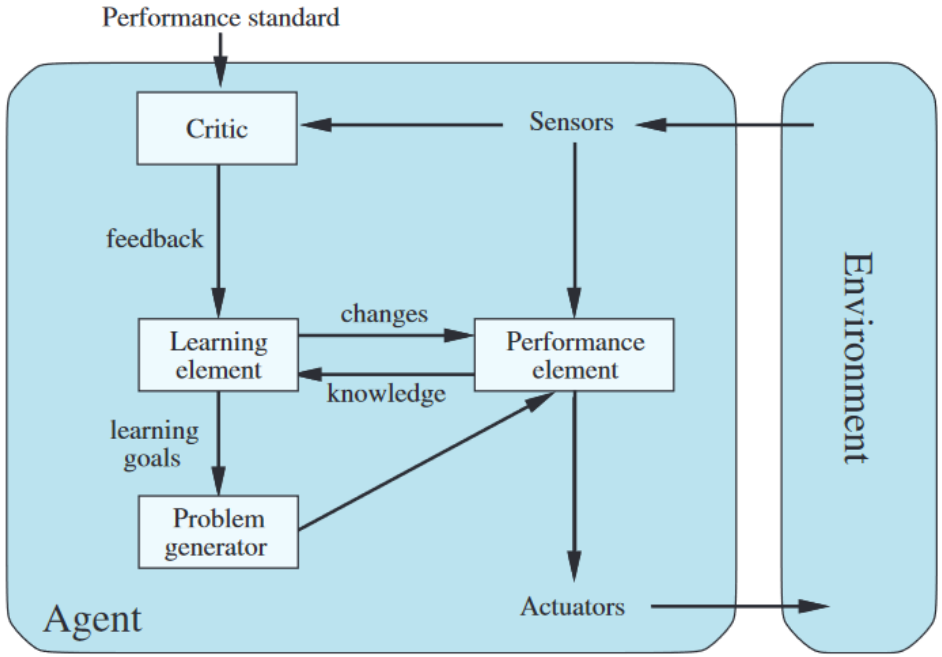
Goals alone are not really enough to generate high-quality behavior. For example, there are many action sequences that will get the taxi to its destination, thereby achieving the goal, but some are quicker, safer, more reliable, or cheaper than others. Goals just provide a crude distinction between "happy" and "unhappy" states, whereas a more general performance measure should allow a comparison of different world states (or sequences of states) according to exactly how happy they would make the agent if they could be achieved. Because "happy" does not sound very scientific, the customary terminology is to say that if one world state is preferred to another, then it has higher utility for the agent.

Utility is therefore a function that maps a state9 onto a real number, which describes the associated degree of happiness.



General learning Agent

A general learning agent. The “performance element” box represents what we have previously considered to be the whole agent program. Now, the “learning element” box gets to modify that program to improve its performance.



Introduction into *Python*

What is Python?

Technische Hochschule Rosenheim

Python ([ˈpʰyːtɔn]) is a general-purpose, commonly interpreted higher programming language.

- **Python** is an interpreted, object-oriented, functional, dynamically-typed, higher programming language with automatic memory management.
- **Python** was developed in February 1991 by Guido van Rossum at the Centrum Wiskunde and Informatica in Amsterdam.
- **Python** is distributed in versions 2.x and 3.x (not backward compatible!).

TIP: Use Python 3 (end of support for Python2 since 2020!).

Note: The name *Python* has nothing to do with the snake, but refers to *Monty Python's Flying Circus*.

Special features of Python

Technische Hochschule Rosenheim

- **Object Orientation**: Everything is an object.
- **Strongly typed**: Each object has a unique immutable type.
- **Dynamically typed**: The type of a variable is decided only at runtime.
- **Automatic memory management**: No manual (error-prone) creation and release of memory.
- **Whitespace sensitive**: Indentation decides about grouping of statements in logical blocks.

Python-Programm

Technische Hochschule Rosenheim

myprogramm.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from argparse import ArgumentParser

calc = {"+" : lambda a, b: a + b,
        "-" : lambda a, b: a - b,
        "*" : lambda a, b: a * b,
        "/" : lambda a, b: a / b,
        "^" : lambda a, b: a ** b}

parser = ArgumentParser()

def do(op):
    if op in calc:
        print("Result: ", calc[op](args.op1, args.op2))
    else:
        parser.error("{} no valid op".format(op))


if __name__ == '__main__':

    parser.add_argument("-o", "--operation", dest="operation", default="plus")
    parser.add_argument("op1", type=float)
    parser.add_argument("op2", type=float)
    args = parser.parse_args()
    op = args.operation

    do(op)
```

REPL

Technische Hochschule Rosenheim

What happens when you call Python without parameters?

```
> python3
```

REPL

Technische Hochschule Rosenheim

What happens when you call Python without parameters?

```
> python3
```

```
Python 3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print(3+2)
5
>>>
```

- Starts Python in REPL (=Read Eval Print Loop) mode
  - *Interactive* or *Language Shell*

Python: Where can I get it?

Technische Hochschule Rosenheim

Python is available on most platforms:

- Windows, Linux/Unix, Mac, ...
- Also for microcontrollers (https://micropython.org/)

Download here:

- Python.org: https://www.python.org/downloads/
- Anaconda: https://www.anaconda.com/
- Miniconda: https://docs.conda.io/en/latest/miniconda.html

Edit Python code

Technische Hochschule Rosenheim

There are several ways to work with *Python*:

- **Shell**: Use the shell in REPL mode
- **Editor**: Visual Studio Code, VIM, Atom, Notepad++, ...
- **IDEs**: PyCharm (Community Edition), Pyzo, ...
- **Online**:
  - https://repl.it/languages/python3
  - http://micropython.org/unicorn/
- **Jupyter Notebooks**: ...later...

Let's get the party started

Technische Hochschule Rosenheim

Variables, assignments and types

The following program example shows definition and usage of simple variables:

```
x = 1    # x verweist aus das 1 Objekt des Typs int
y = 1.0  # y verweist auf das 1.0 Objekt des Typs float

print("x =", x, "has  the  Type",type(x))
print("y =", y, "has  the  Type",type(y))

x = x + 3
y = y * 2
print(x)
print(y)
```

https://repl.it/@marceltilly/TH-Rosenheim#lecture/01-variables.py

Control structures 101

Technische Hochschule Rosenheim

IF

```
#  condition  if
condition = "" or (3 - 3)
if condition:
    print("Condition  is true!")
elif 1 == 2:
    print("1 is  equal to 2!")
else:
    print("Nothing  true  here :(")
```

https://repl.it/@marceltilly/TH-Rosenheim#lecture/06-if.py

Exercise 3

Technische Hochschule Rosenheim

Write your first Python programm

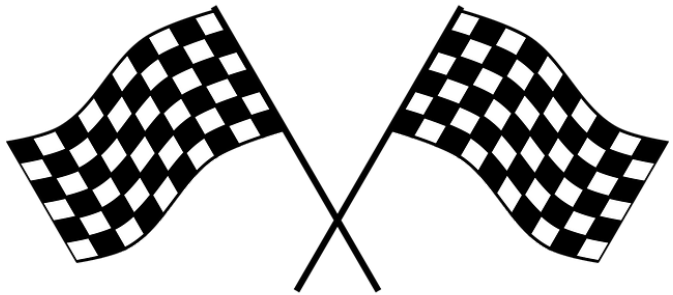
Given two numbers, write a Python code to find the Maximum of these two numbers.

Examples:

```
Input: a = 2, b = 4
Output: 4

Input: a = -1, b = -4
Output: -1
```

Summary

- Agents
- Performance Measure
- Environments
- PEAS
- All agents improve their perfromacne through **learning**
- Introduction into Python



Homework

- For each of the following activities, give a PEAS description of the task environment and characterize it in terms of the properties listed above>
  - Playing soccer.
  - Shopping for used AI books on the Internet.
  - Playing a tennis match.
- Write pseudocode agent programs for the goal-based and utility-based agents.

References

- Norvig, Russel: AI: A Modern Approoach: http://aima.cs.berkeley.edu/
- Entertainment: https://www.youtube.com/watch?v=SSE4M0gcmvE
- We need to talk AI: https://weneedtotalk.ai/

When AI meets GIT!