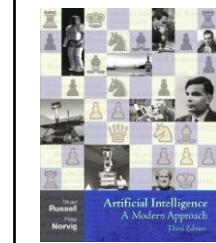


# Agents

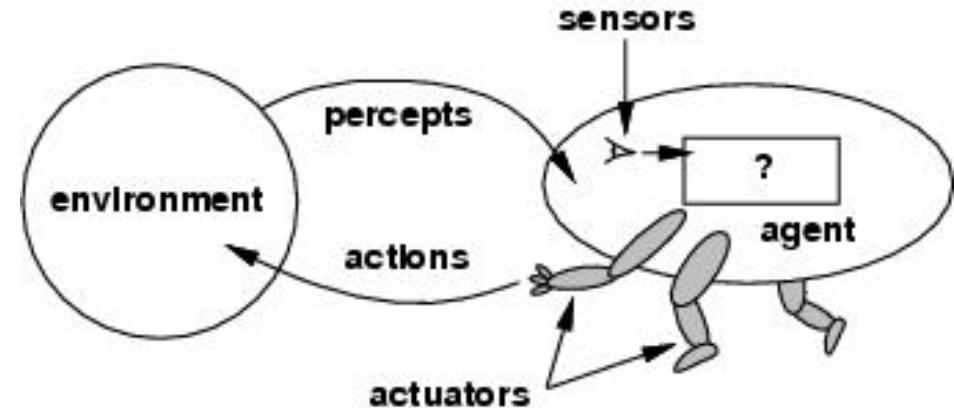
- This course is about designing intelligent agents
- Agents and environments
  - The vacuum-cleaner world
- Rationality
  - The concept of rational behavior.
- Environment types
- Agent types



Many slides based on  
Russell & Norvig's slides  
[Artificial Intelligence:  
A Modern Approach](#)

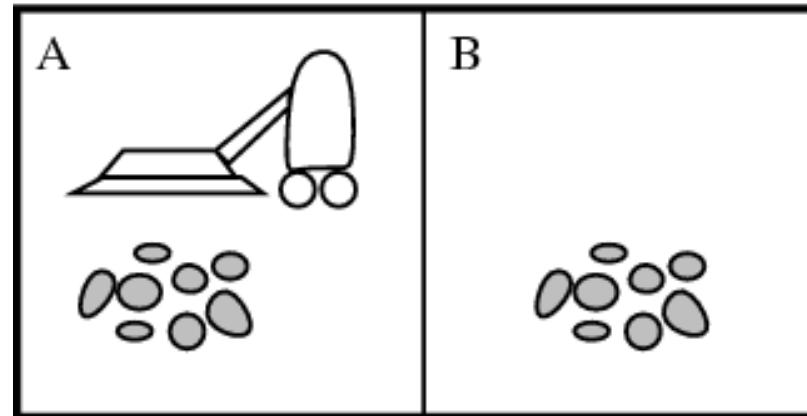
# Agents

- An **agent** is an entity that perceives and acts in an environment
  - environment can be real or virtual
- An agent can always perceive its actions, but not necessarily their effects on the environment
- **Rational agent:** optimizes some performance criterion
  - For any given task and class of environments we seek the agent (or class of agents) with the best performance.
- Problem:
  - computational limitations make perfect rationality unachievable



# The Vacuum-Cleaner world

- A robot-vacuum-cleaner that operates in a simple world



- **Environment:**
  - Virtual house with room A and room B
- **Percepts:**
  - The robot can sense pairs [ $<location>$ ,  $<status>$ ]
    - *Location*: whether it is in room *A* or *B*
    - *Status*: whether the room is *Clean* or *Dirty*
- **Actions:**
  - *Left*, *Right*, *Suck*, *NoOp*

# Rational Agent – Performance Measure

- A **rational agent** is an agent that “does the right thing”
  - intuitively clear, but needs to be measurable in order to be useful for computer implementation
- **Performance Measure:**
  - a function that evaluates sequence of actions/environment states
  - obviously not fixed but task-dependent
- Vacuum-World performance measures:
  - reward for the amount of dust cleaned
    - one point per square cleaned up in time  $T$
    - can be maximized by dumping dust on the floor again...
  - reward for clean floors
    - one point per clean square per time step
  - possibly with penalty for consumed energy
    - minus one per move?
- General rule:
  - design performance measure based on desired environment state
  - not on desired agent behavior

# Rational Agent

*A rational agent chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date and prior environment knowledge.*

- Rational ≠ omniscient
  - An omniscient agent knows the actual outcome of its actions.
- Rational ≠ successful
  - Rationality maximizes expected performance
  - This may not be the optimal outcome
  - Example:
    - the expected monetary outcome of playing in the lottery/casino, etc. is negative (hence it is rational not to play)
    - but if you're lucky, you may win...

# PEAS

What is rational at a given time depends on four things:

**P**: the **performance measure** that defines the success

**E**: the agent's prior knowledge of the **environment**

**A**: the **actions** that the agent can perform

**S**: the agent's **percept sequence** to date

- *Example: Fully automated Taxi*



# PEAS

What is rational at a given time depends on four things:

**P**: the **performance measure** that defines the success

**E**: the agent's prior knowledge of the **environment**

**A**: the **actions** that the agent can perform

**S**: the agent's **percept sequence** to date

- *Example: Fully automated Taxi*

- **Performance**

- Safety, destination, profits, legality, comfort

- **Environment**

- Streets/freeways, other traffic, pedestrians, weather, ...

- **Actuators**

- Steering, accelerating, brake, horn, speaker/display,...

- **Sensors**

- Video, sonar, speedometer, engine sensors, keyboard, GPS, ...



# PEAS

What is rational at a given time depends on four things:

**P**: the **performance measure** that defines the success

**E**: the agent's prior knowledge of the **environment**

**A**: the **actions** that the agent can perform

**S**: the agent's **percept sequence** to date

- *Example: Internet Shopping Agent*



# PEAS

What is rational at a given time depends on four things:

**P**: the **performance measure** that defines the success

**E**: the agent's prior knowledge of the **environment**

**A**: the **actions** that the agent can perform

**S**: the agent's **percept sequence** to date

- *Example: Internet Shopping Agent*

- **Performance**
  - price, quality, appropriateness, efficiency
- **Environment**
  - the Web: current and future WWW sites, vendors, shippers
- **Actuators**
  - display to user, follow URL, fill in form
- **Sensors**
  - parsing of HTML pages (text, graphics, scripts)...



# PEAS

What is rational at a given time depends on four things:

**P**: the **performance measure** that defines the success

**E**: the agent's prior knowledge of the **environment**

**A**: the **actions** that the agent can perform

**S**: the agent's **percept sequence** to date

- *Example: Chess Program*



# PEAS

What is rational at a given time depends on four things:

**P**: the **performance measure** that defines the success

**E**: the agent's prior knowledge of the **environment**

**A**: the **actions** that the agent can perform

**S**: the agent's **percept sequence** to date

- *Example: Chess Program*

- **Performance**

- number of games won, ELO rating,...

- **Environment**

- the chess board

- **Actuators**

- moves that can be performed

- **Sensors**

- placement of pieces in current position, whose turn is it?, ...



# Environment Types

- *Fully observable*
  - the complete state of the environment can be sensed
  - at least the relevant parts
  - no need to keep track of internal states
- *Partially observable*
  - parts of the environment cannot be sensed

Task Environment	Observable
Sudoku	
Chess With a Clock	
Poker	
Backgammon	
Taxi driving	
Medical diagnosis	
Image Analysis	
Part-Picking Robot	
Refinery Controller	
Interactive Tutor	

# Environment Types

- *Deterministic*
  - the next environment state is completely determined by the current state and the executed action
- *Strategic*
  - only the opponents' actions cannot be foreseen. That is, the environment is deterministic except for the actions of other agents
- *Stochastic*

Task Environment	Observable	Deterministic
Sudoku	<i>Fully</i>	
Chess With a Clock	<i>Fully</i>	
Poker	<i>Partially</i>	
Backgammon	<i>Fully</i>	
Taxi driving	<i>Partially</i>	
Medical diagnosis	<i>Partially</i>	
Image Analysis	<i>Fully</i>	
Part-Picking Robot	<i>Partially</i>	
Refinery Controller	<i>Partially</i>	
Interactive Tutor	<i>Partially</i>	

# Environment Types

- *Episodic*
  - the agent's experience can be divided into atomic steps
  - the agent perceives and then performs a single action
  - the choice of action depends only on the episode itself
- *Sequential*
  - the current decision could influence all future decision

Task Environment	Observable	Deterministic	Episodic
Sudoku	Fully	Deterministic	
Chess With a Clock	Fully	Strategic	
Poker	Partially	Strategic	
Backgammon	Fully	Stochastic	
Taxi driving	Partially	Stochastic	
Medical diagnosis	Partially	Stochastic	
Image Analysis	Fully	Deterministic	
Part-Picking Robot	Partially	Stochastic	
Refinery Controller	Partially	Stochastic	
Interactive Tutor	Partially	Stochastic	

# Environment Types

- *Dynamic*
  - the environment may change while the agent deliberates
- *Static*
  - the environment does not change while the agent deliberates
- *Semidynamic*
  - the environment does not change, but the performance score may

Task Environment	Observable	Deterministic	Episodic	Static
Sudoku	Fully	Deterministic	Sequential	
Chess With a Clock	Fully	Strategic	Sequential	
Poker	Partially	Strategic	Sequential	
Backgammon	Fully	Stochastic	Sequential	
Taxi driving	Partially	Stochastic	Sequential	
Medical diagnosis	Partially	Stochastic	Sequential	
Image Analysis	Fully	Deterministic	Episodic	
Part-Picking Robot	Partially	Stochastic	Episodic	
Refinery Controller	Partially	Stochastic	Sequential	
Interactive Tutor	Partially	Stochastic	Sequential	

# Environment Types

- *Discrete*
  - finite number of actions / environment states / percepts
- *Continuous*
  - actions, states, percepts are on a continuous scale
- this distinction applies separately to actions, states, and percepts
  - can be mixed in individual tasks

<b>Task Environment</b>	<b>Observable</b>	<b>Deterministic</b>	<b>Episodic</b>	<b>Static</b>	<b>Discrete</b>
Sudoku	<i>Fully</i>	<i>Deterministic</i>	<i>Sequential</i>	<i>Static</i>	
Chess With a Clock	<i>Fully</i>	<i>Strategic</i>	<i>Sequential</i>	<i>Semi</i>	
Poker	<i>Partially</i>	<i>Strategic</i>	<i>Sequential</i>	<i>Static</i>	
Backgammon	<i>Fully</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Static</i>	
Taxi driving	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	
Medical diagnosis	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	
Image Analysis	<i>Fully</i>	<i>Deterministic</i>	<i>Episodic</i>	<i>Semi</i>	
Part-Picking Robot	<i>Partially</i>	<i>Stochastic</i>	<i>Episodic</i>	<i>Dynamic</i>	
Refinery Controller	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	
Interactive Tutor	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	

# Environment Types

- *Single-Agent*
  - No other agents (other agents may be part of the environment)
- *Multi-Agent*
  - Does the environment contain other agents whose performance measure depends on my *actions*?
  - other agents may be *co-operative* or *competitive*

<b>Task Environment</b>	<b>Observable</b>	<b>Deterministic</b>	<b>Episodic</b>	<b>Static</b>	<b>Discrete</b>	<b>Agents</b>
Sudoku	<i>Fully</i>	<i>Deterministic</i>	<i>Sequential</i>	<i>Static</i>	<i>Discrete</i>	
Chess With a Clock	<i>Fully</i>	<i>Strategic</i>	<i>Sequential</i>	<i>Semi</i>	<i>Discrete</i>	
Poker	<i>Partially</i>	<i>Strategic</i>	<i>Sequential</i>	<i>Static</i>	<i>Discrete</i>	
Backgammon	<i>Fully</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Static</i>	<i>Discrete</i>	
Taxi driving	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	<i>Continuous</i>	
Medical diagnosis	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	<i>Continuous</i>	
Image Analysis	<i>Fully</i>	<i>Deterministic</i>	<i>Episodic</i>	<i>Semi</i>	<i>Continuous</i>	
Part-Picking Robot	<i>Partially</i>	<i>Stochastic</i>	<i>Episodic</i>	<i>Dynamic</i>	<i>Continuous</i>	
Refinery Controller	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	<i>Continuous</i>	
Interactive Tutor	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	<i>Discrete</i>	

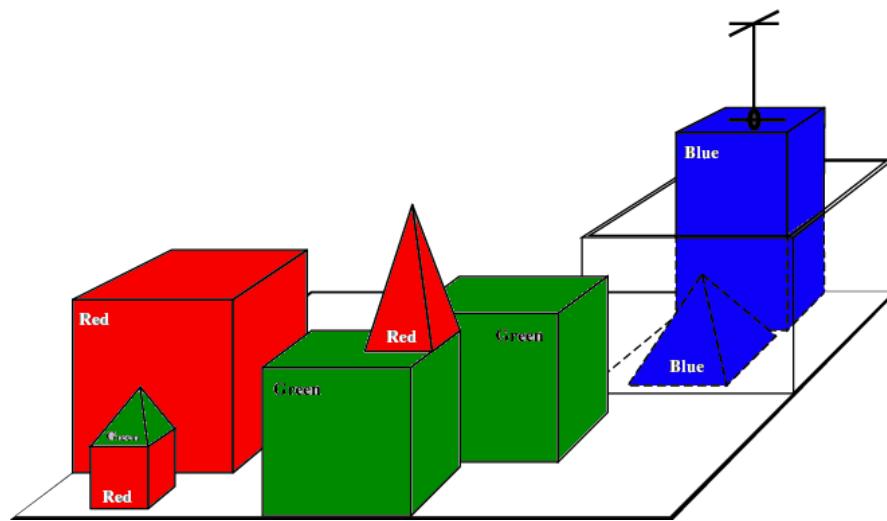
# Environment Types

- *Single-Agent*
  - No other agents (other agents may be part of the environment)
- *Multi-Agent*
  - Does the environment contain other agents whose performance measure depends on my *actions*?
  - other agents may be *co-operative* or *competitive*

<b>Task Environment</b>	<b>Observable</b>	<b>Deterministic</b>	<b>Episodic</b>	<b>Static</b>	<b>Discrete</b>	<b>Agents</b>
Sudoku	<i>Fully</i>	<i>Deterministic</i>	<i>Sequential</i>	<i>Static</i>	<i>Discrete</i>	<i>Single</i>
Chess With a Clock	<i>Fully</i>	<i>Strategic</i>	<i>Sequential</i>	<i>Semi</i>	<i>Discrete</i>	<i>Multi</i>
Poker	<i>Partially</i>	<i>Strategic</i>	<i>Sequential</i>	<i>Static</i>	<i>Discrete</i>	<i>Multi</i>
Backgammon	<i>Fully</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Static</i>	<i>Discrete</i>	<i>Multi</i>
Taxi driving	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	<i>Continuous</i>	<i>Multi</i>
Medical diagnosis	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	<i>Continuous</i>	<i>Single</i>
Image Analysis	<i>Fully</i>	<i>Deterministic</i>	<i>Episodic</i>	<i>Semi</i>	<i>Continuous</i>	<i>Single</i>
Part-Picking Robot	<i>Partially</i>	<i>Stochastic</i>	<i>Episodic</i>	<i>Dynamic</i>	<i>Continuous</i>	<i>Single</i>
Refinery Controller	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	<i>Continuous</i>	<i>Single</i>
Interactive Tutor	<i>Partially</i>	<i>Stochastic</i>	<i>Sequential</i>	<i>Dynamic</i>	<i>Discrete</i>	<i>Multi</i>

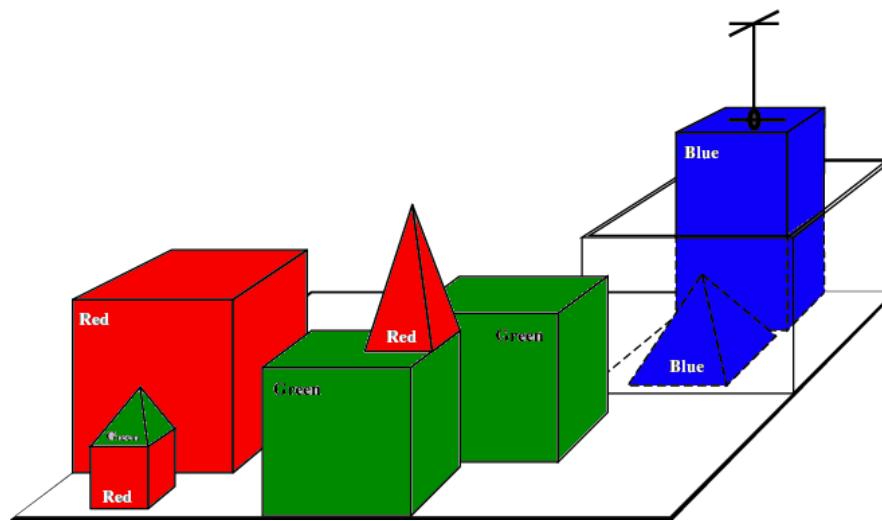
# Environment Types

- The simplest environment is
  - fully observable
  - deterministic
  - episodic
  - static
  - discrete
  - single-agent



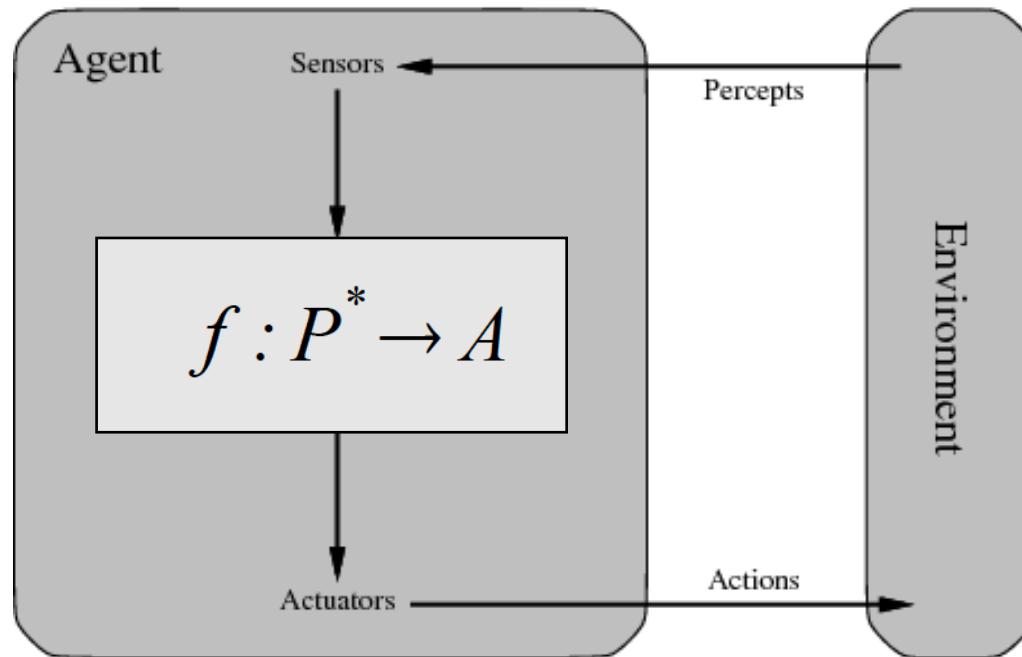
# Environment Types

- The simplest environment is
  - fully observable
  - deterministic
  - episodic
  - static
  - discrete
  - single-agent
- Most real situations are
  - partially observable
  - stochastic
  - sequential
  - dynamic
  - continuous
  - multi-agent



# Agent Function

- The agent function maps percept histories to actions



- The agent function will internally be represented by the agent program.
- The agent program runs on the physical architecture to produce  $f$ .

# A Simple Vacuum Cleaner Agent

- Table of mappings from Percept sequences to actions:

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 Simple General Agent

```
function TABLE-DRIVEN-AGENT(percept) returns an action
    static: percepts, a sequence initially empty
            table, a table of actions, indexed by percept sequence
    append percept to the end of percepts
    action  $\leftarrow$  LOOKUP(percepts, table)
    return action
```

- has a table of all possible percept histories
- looks up the right response in the table
- Clearly infeasible:
  - if there are  $|P|$  percepts and a life-time of  $T$  time steps,  
we need a look-up table of size  $\sum_{t=1}^T |P^t|$
- For example: chess:
  - about 36 moves per position, average game-length 40 moves  
 $\rightarrow 5105426007029058700898070779698222806522450657188621232590965$

# A Simple Vacuum Cleaner Agent

- Strategy

“If current room is dirty then suck, otherwise move to the other room.”

- As an agent program

```
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
```

- Obvious Questions:

- Is this the right agent?
- Is this a good agent?
- Is there a right agent?

# Agent Programs

*The key challenge for AI is to write programs that produce rational behavior from a small amount of code rather than a large number of table entries*

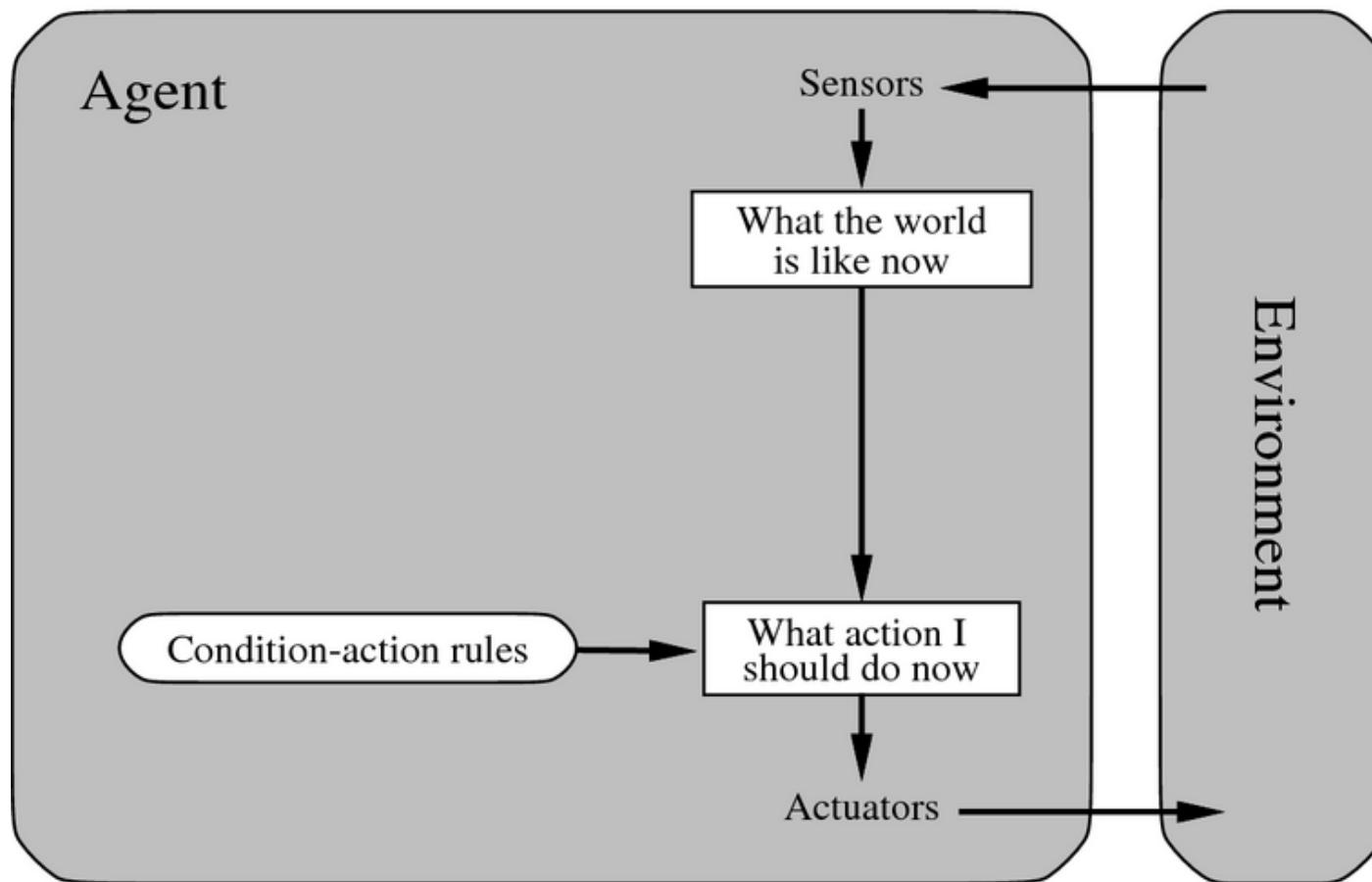
- Writing down the agent functions is not practical for real applications
- But feasibility is also important
  - you can write a perfect chess playing agent with a few lines of code
  - it will run forever, though...
- Agent = architecture + program

# Agent Types

- Four basic kinds of agent programs will be discussed:
  - Simple reflex agents
  - Model-based reflex agents
  - Goal-based agents
  - Utility-based agents
- All these can be turned into learning agents.

# Simple Reflex Agent

- Select action on the basis of only the current percept
  - ignores the percept history



# Simple Reflex Agent

- Select action on the basis of only the current percept
  - ignores the percept history
- Implemented through condition-action rules
- Large reduction in possible percept/action situations
  - from  $\sum_{t=1}^T |P^t|$  to  $|P|$
- But will make a very bad chess player
  - does not look at the board, only at the opponent's last move  
(assuming that the sensory input is only the last move, no visual)

*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
```

# General Simple Reflex Agent

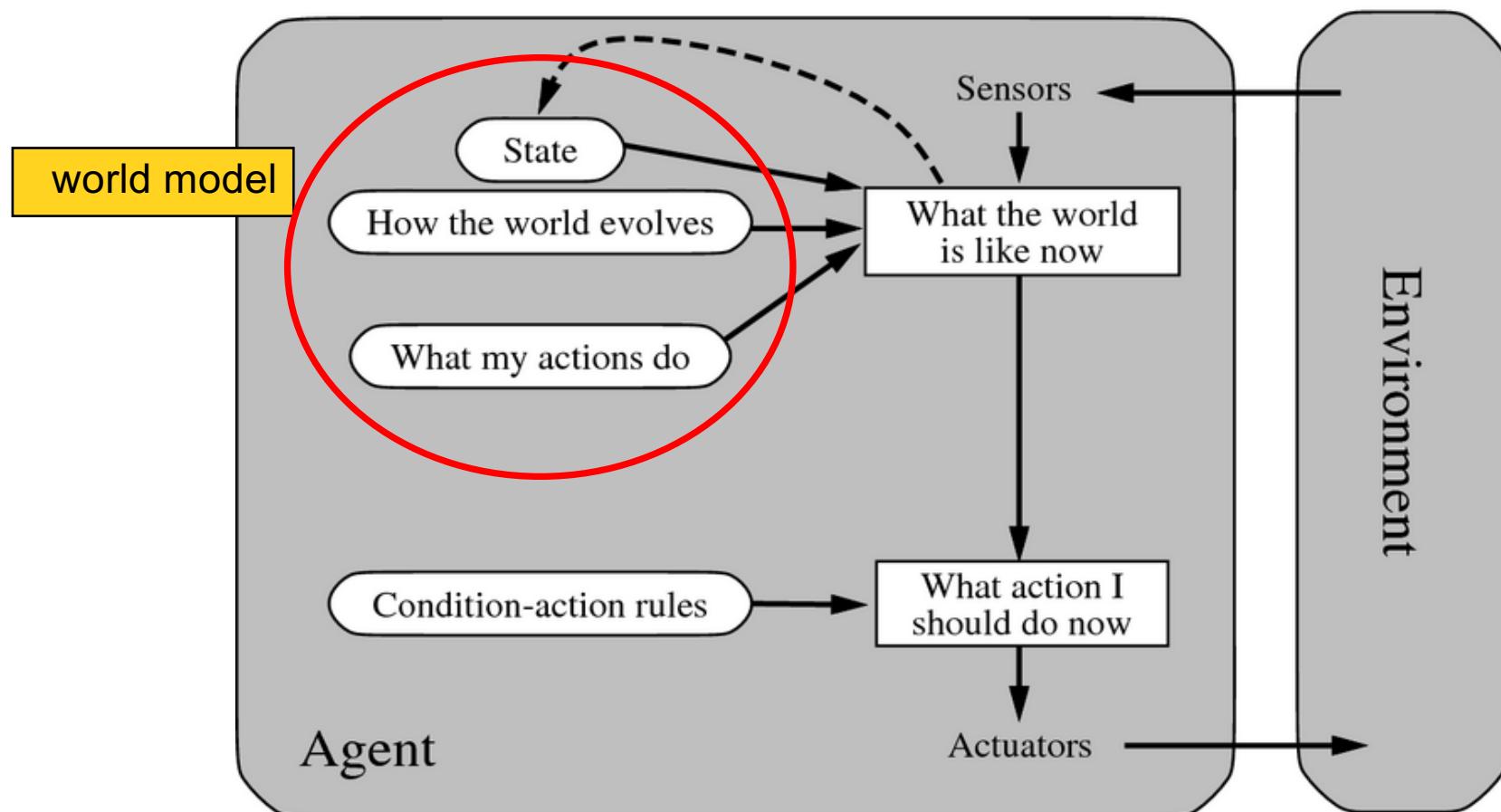
```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  static: rules, a set of condition-action rules

    state  $\leftarrow$  INTERPRET-INPUT(percept)
    rule  $\leftarrow$  RULE-MATCH(state, rule)
    action  $\leftarrow$  RULE-ACTION[rule]
    return action
```

- Note that rules are just used as a concept
  - actual implementation could, e.g., be logical circuitry
- Will only work if the environment is *fully observable*
  - everything important needs to be determinable from the current sensory input
  - otherwise infinite loops may occur
    - e.g. in the vacuum world without a sensor for the room, the agent does not know whether to move right or left
    - possible solution: randomization

# Model-Based Reflex Agent

- Keep track of the state of the world
  - better way to fight partial observability



# General Model-Based Reflex Agent

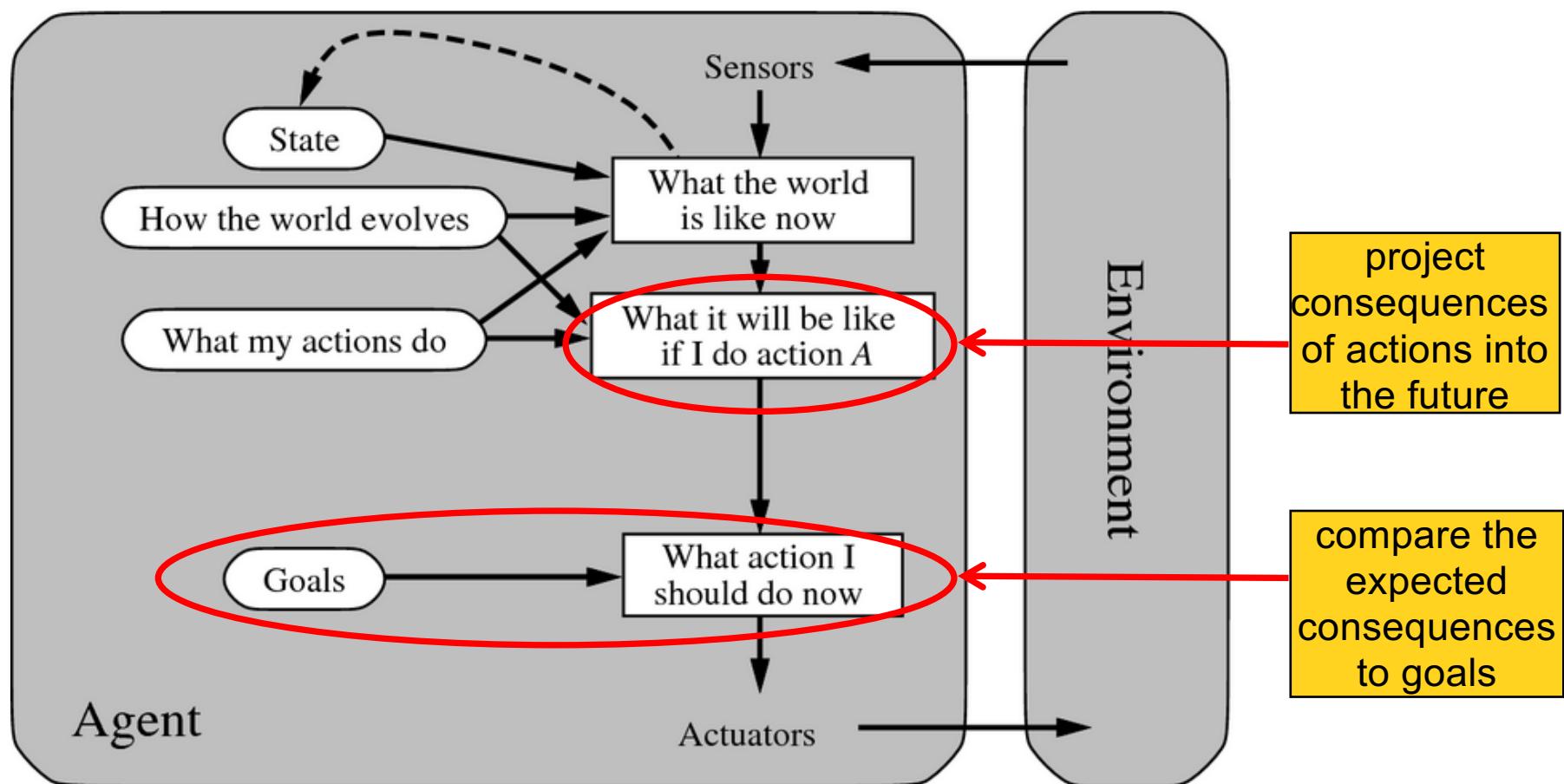
```
function REFLEX-AGENT-WITH-STATE(percept) returns an action
    static: state, a description of the current world state
            rules, a set of condition-action rules
            action, the most recent action, initially none

    state  $\leftarrow$  UPDATE-STATE(state, action, percept)
    rule  $\leftarrow$  RULE-MATCH(state, rule)
    action  $\leftarrow$  RULE-ACTION[rule]
    return action
```

- Input is not only interpreted, but mapped into an internal state description (a world model)
  - a chess agent could keep track of the current board situation when its percepts are only the moves
- Internal state is also used for interpreting subsequent percepts
- The world model may include effects of own actions!

# Goal-Based Agent

- the agent knows what states are desirable
  - it will try to choose an action that leads to a desirable state

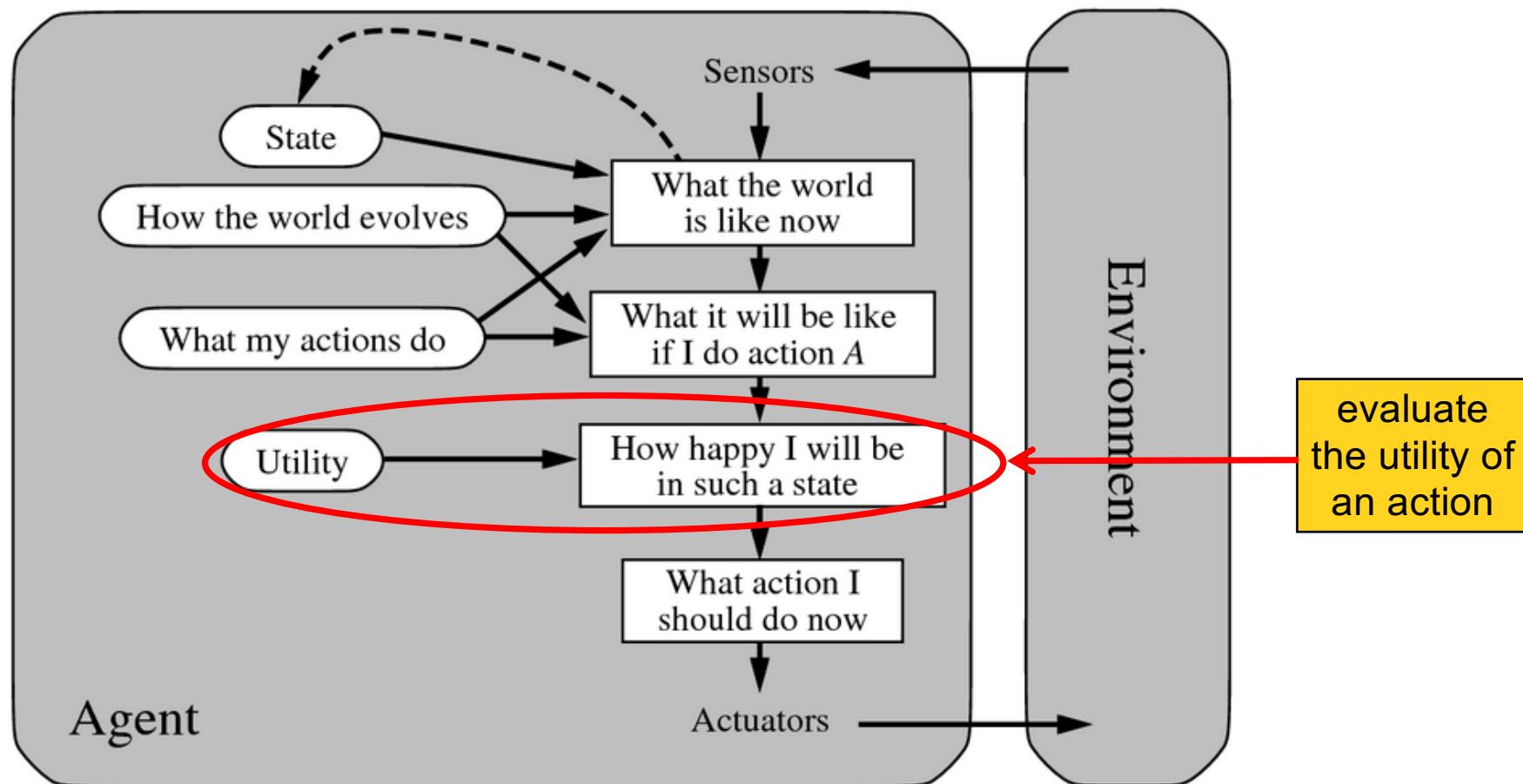


# Goal-Based Agent

- the agent knows what states are desirable
  - it will try to choose an action that leads to a desirable state
- things become difficult when long sequences of actions are required to find the goal
  - typically investigated in search and planning research
- main difference to previous approaches
  - decision-making takes future into account
    - “What will happen if I do such-and-such?”
    - “Will this make me happy?”
- is more flexible since knowledge is represented explicitly and can be manipulated
  - changing the goal does not imply changing the entire set of condition-action rules

# Utility-Based Agent

- Goals provide just a binary happy/unhappy distinction
  - utility functions provide a continuous scale



# Utility-Based Agent

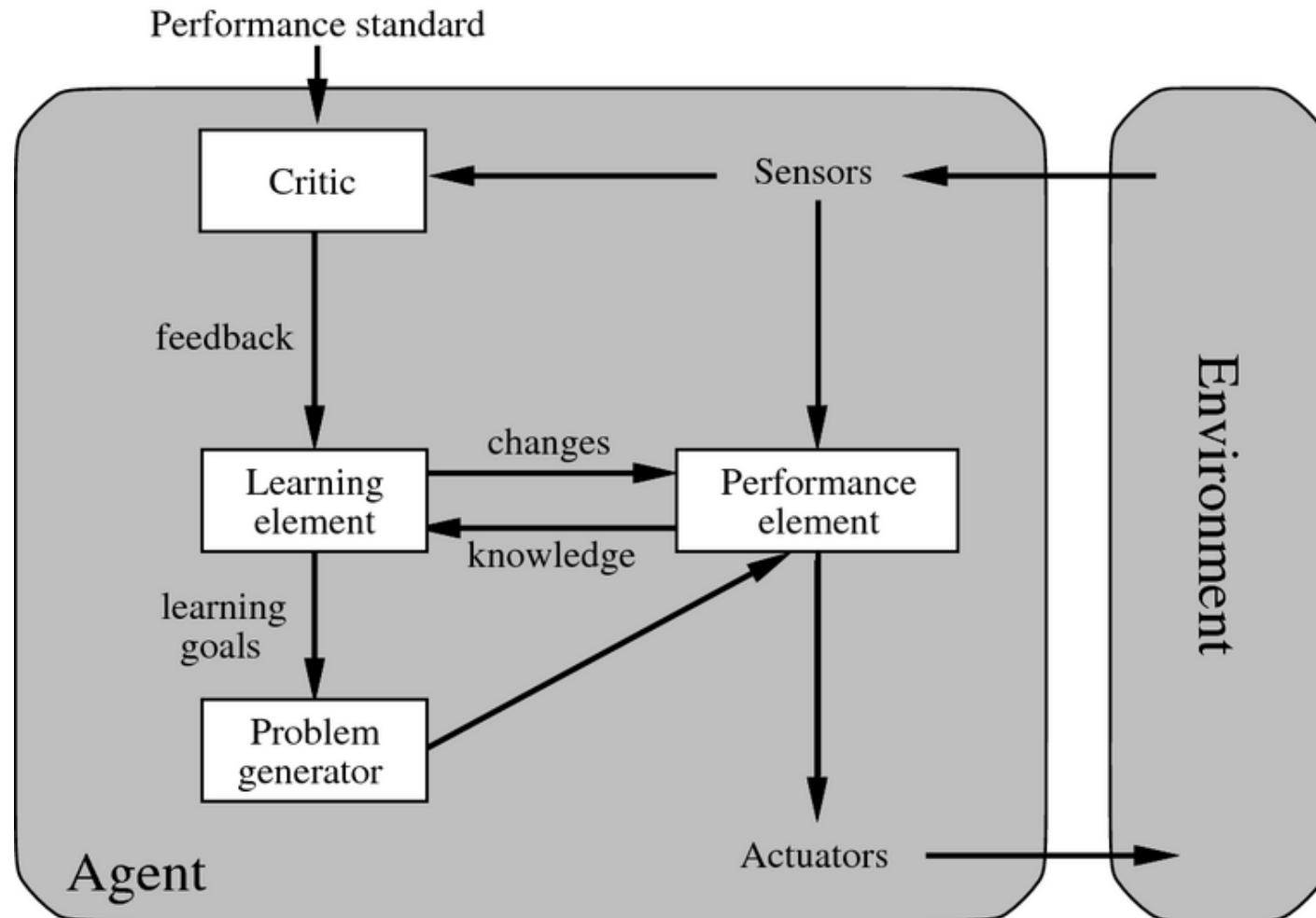
- Goals provide just a binary happy/unhappy distinction
  - utility functions provide a continuous scale
- Certain goals can be reached in different ways.
  - “Alle Wege führen nach Rom”
  - Some ways are quicker, safer, more reliable, cheaper, ...
    - → have a higher utility
- Utility function
  - maps a state (or a sequence of states) onto a real number
- Improves on goals:
  - selection between conflicting goals (e.g., speed and safety)
  - selection between goals based on trade-off between likelihood of success and importance of goal

# Learning

- All previous agent-programs describe methods for selecting actions
  - yet they do not explain the origin of these programs.
- Learning mechanisms can be used for acquiring programs
  - teach them instead of instructing them
  - but you still have to program the learning mechanism!
- Advantage
  - robustness of the program toward initially unknown environments.
- Every part of the previous agents can be improved with learning, but proving things about the learned program might hard(er)

*Learning in intelligent agents can be summarized as a process of modification of each component of the agent to bring the components into closer agreement with the available feedback information, thereby improving the overall performance of the agent.*

# Learning Agent



# Learning Agent

- **Performance element**
  - makes the action selection (as usual)
- **Critic**
  - decides how well the learner is doing with respect to a fixed *performance standard*
  - necessary because the percepts do not provide any indication of the agent's success
    - e.g., it needs to know that checkmate is bad
- **Learning element**
  - improves the performance element
  - its design depends very much on the performance element
- **Problem generator**
  - responsible for *exploration* of new knowledge
    - sometimes try new, possibly suboptimal actions to acquire knowledge about their consequences
  - otherwise only *exploitation* of (insufficient) current knowledges

# Big Picture

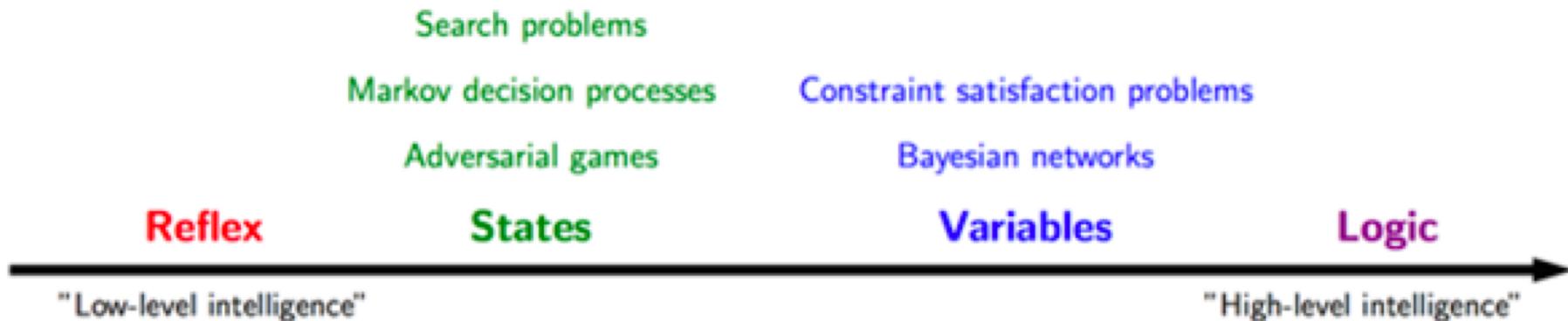
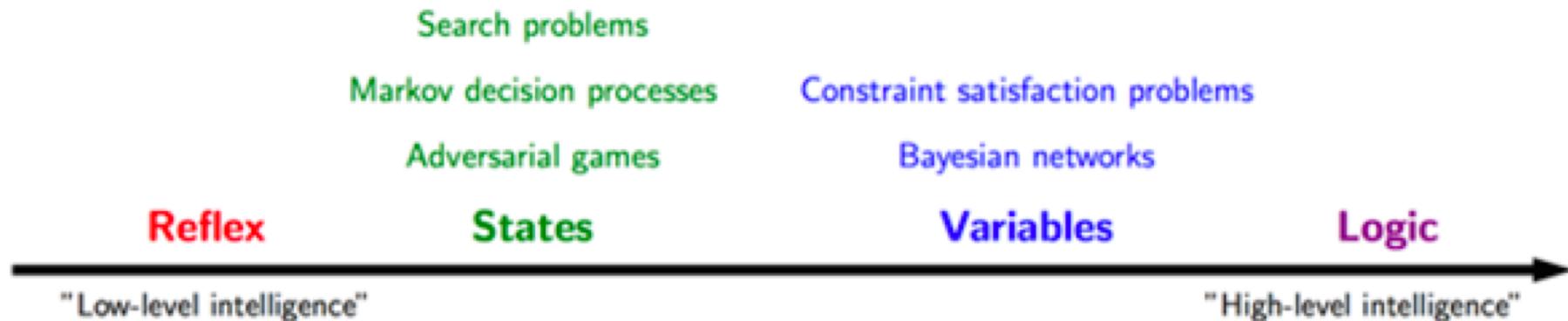


Image due to Percy Liang

As the agents get complex, so does their internal structure. The way in which they store the internal state changes. By its nature, a simple reflex agent does not need to store a state, but other types do.

# Atomic representation

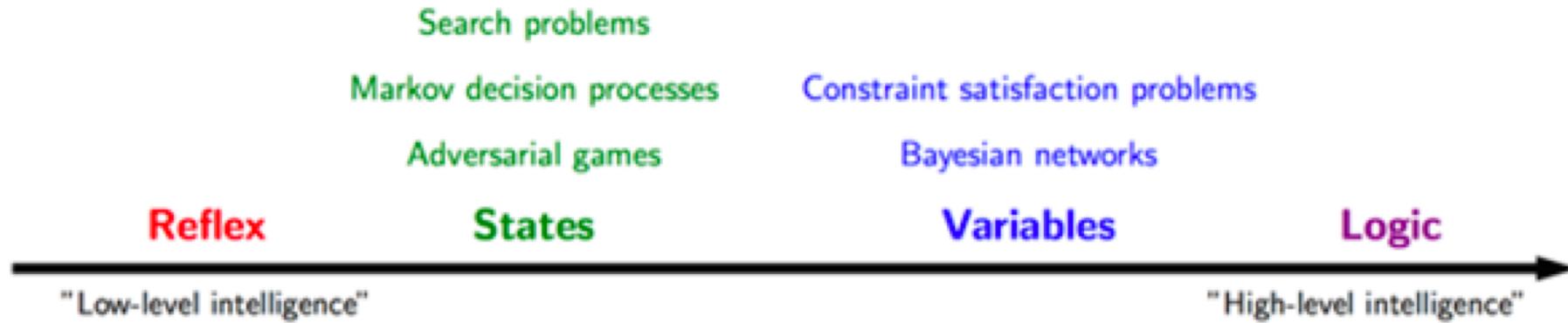


***State is stored as black box, i.e. without any internal structure.***

For example, for Roomba(a robotic vaccum cleaner), the internal state is a patch already vaccumed, you don't have to know anything else. As depicted in the image, such representation works for model and goal based agents and ***used in various AI algorithms such as search problems and adversarial games.***

Image due to Percy Liang

# Factored representation

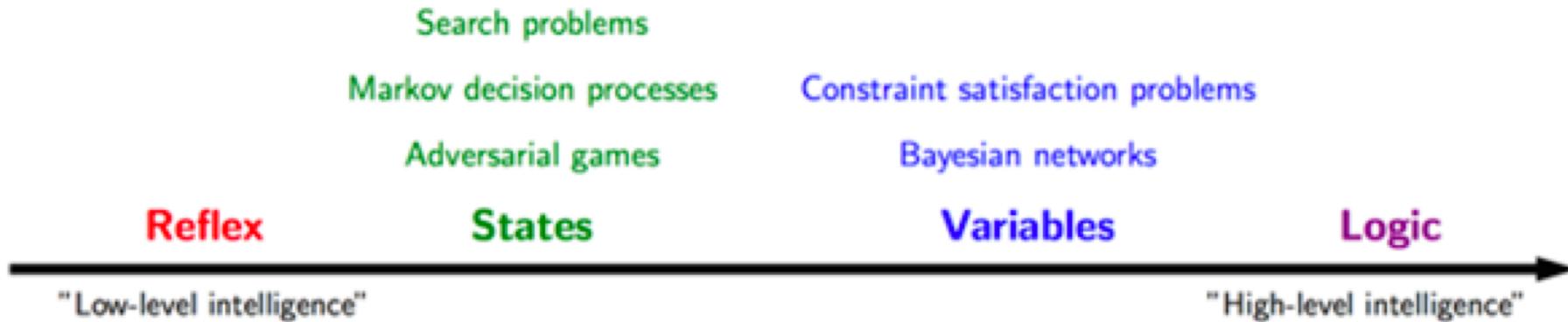


**State is no longer a black box. *It now has attribute-value pairs, also known as variables that can contain a value.***

For example, while finding a route, you have a GPS location and amount of gas in the tank. This adds a constraint to the problem. Factored representation works for goal based agents and are *used in various AI algorithms such as constraint satisfaction and bayesian networks, even in deep networks*

Image due to Percy Liang

# Structured representation



**Now, we have relationships between the variables/factored states. This induces logic in the AI algorithms.**

For example, in natural language processing, the states are whether the statement contains a reference to a person and whether the adjective in that statement represents that person. The relation in these states will decide, whether the statement was a sarcastic one. ***This is high level Artificial Intelligence, used in algorithms like first order logic, knowledge-based learning and natural language understanding. But also in deep networks***

Image due to Percy Liang