Artificial Intelligence

2. Intelligent Agents

Shashi Prabh

School of Engineering and Applied Science

Ahmedabad University

Contents

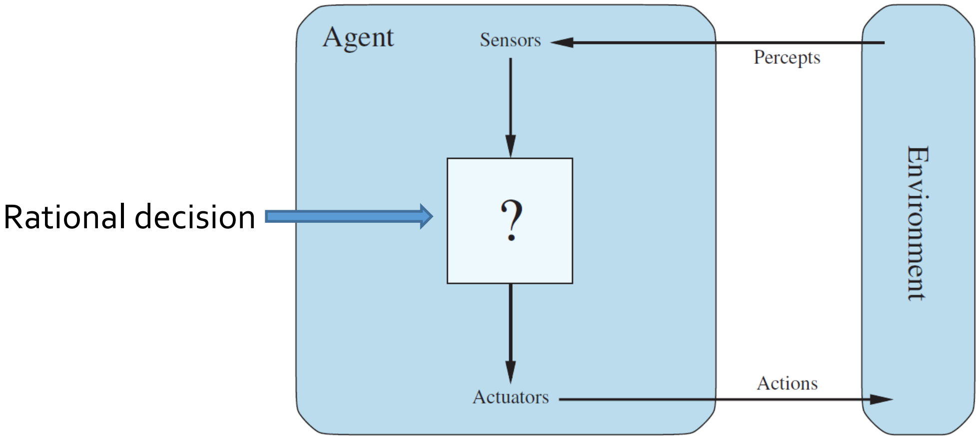
Goal: use the concept of rationality to develop a set of design principles for building AI systems a.k.a. intelligent agents

Topics

- Agents
- Rationality
- Environment characterization
- Agent types

Agents

- Agent is an entity that perceives its environment through sensors and acts upon that environment through actuators
  - Significant computational resources, complex environment, non-trivial decisions
  - Just a tool for analysing systems



Agents interact with environments through sensors and actuators.

Agent examples

Agent	Sensors	Actuators
Humans	Eyes, ears, skin	Hands, legs, vocal chord
Robots	Camera, sound sensor, IR range finder	Motors, pumps, drills, displays
Software	File content, human input, packets received over network	Write file, display/print information, Send packets over network

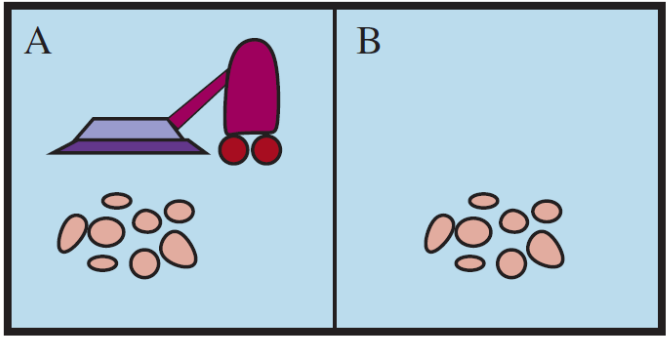
Agents

- **Environment:** part of the universe whose state is relevant for designing the agent

- **Percept:** content sensed by the agent's sensors
- **Percept sequence:** complete history of an agent's percepts
  - Determines the choice of actions

- **Agent function** (f): mapping of a percept sequence to an action
  - Determines an agent's behavior
- **Agent program:** Implementation of f

Vacuum-World Example



- Word consists of 2 squares, A and B
- Percepts available
  - location (A or B)
  - Is the room dirty?
- Actions available
  - Move to left or right
  - Suck up the dirt
  - Do nothing
- Agent function: if the square is dirty, then suck. Otherwise move to the other square
  - Is it a good agent design?

Vacuum-World Agent Function

Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Clean], [A, Dirty]$	$Suck$
⋮	⋮

- Specifying the actions differently leads to various vacuum-world agents
  - What is the right way?

Rationality: Performance Measure

- What makes an agent good / bad / stupid?
- A rational agent is supposed to do the "right thing"
  - What does "right" mean? How to define it?

- Consequentialism as performance measure: behaviour is evaluated by its consequences
  - As the result of actions, environment transitions through a sequence of states
  - A consequentialist performance measure evaluates an agent based on the sequence of environment states
- Performance measure needs to be specified by the designer/user
  - Machines don't have innate preferences or desires
  - It is usually hard to formulate it

Rationality: Performance Measure Example

Vacuum-World

- Option 1: The amount of dirt cleaned-up in a day
  - Bad idea. Why?
- Option 2: Is the floor clean?
  - + 2 for clean floor (every minute), -3 for dirty floor, -1 for noise
- Weiner: "the purpose to put into the machine is the purpose we really desire."
- General guideline: Performance measure should reflect what one wants to achieve not how the agent should behave (consequentialism…)

Rationality

- Rationality at any given time depends on
  - Performance measure (the criterion of success)
  - Agent's prior knowledge of the environment
  - Available actions
  - Percept sequence
- Rational agent: take actions that are expected to maximize performance measure
  - The characteristics of sensors, actuators and environment dictate techniques for selecting actions
- **Rationality ≠ Omniscience**
  - Do not expect the agent to do what turns after the fact to be the best action

Nature of Environments

- Task environments are the problems of which rational agents are the solutions
- Task environment specification : Performance, Environment, Actuators, Sensors (PEAS)
- PEAS description of self-driving car task environment

Nature of Environments

- Task environments are the problems of which rational agents are the solutions
- Task environment specification : Performance, Environment, Actuators, Sensors (PEAS)
- PEAS description of self-driving car task environment

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits, minimize impact on other road users	Roads, other traffic, police, pedestrians, customers, weather	Steering, accelerator, brake, signal, horn, display, speech	Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen

Examples

- PEAS examples

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

Properties of Task Environments

Task environments vary along several significant dimensions:

1. fully or partially observable
2. single-agent or multi-agent
3. deterministic or nondeterministic
4. episodic or sequential
5. static or dynamic
6. discrete or continuous
7. known or unknown

Properties of Task Environments

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

Structure of Agents

- Agent program : implementation of agent function
  - Takes percept sequence as input
  - Outputs action
- Agent architecture : composition of physical device(s) running the agent program
  - Sensors
  - Actuators
  - Computing device
- **Agent = Program + Architecture**

Agent Program : Table-Driven Agent

- Keeps track of percept sequence
- Maintains a mapping between percept sequence and action in a table
- Looks-up in the table to find the action
- Not practical - Table can get very large
  - 10^{150} entries for a chess program, 10^{80} atoms in the universe

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-Driven Agent

```
def TableDrivenVacuumAgent():
    """Tabular approach towards vacuum world as mentioned in [Figure 2.3]
    >>> agent = TableDrivenVacuumAgent()
    >>> environment = TrivialVacuumEnvironment()
    >>> environment.add_thing(agent)
    >>> environment.run()
    >>> environment.status == {(1,0):'Clean' , (0,0) : 'Clean'}
    True
    """
    table = {((loc_A, 'Clean'),): 'Right',
             ((loc_A, 'Dirty'),): 'Suck',
             ((loc_B, 'Clean'),): 'Left',
             ((loc_B, 'Dirty'),): 'Suck',
             ((loc_A, 'Dirty'), (loc_A, 'Clean')): 'Right',
             ((loc_A, 'Clean'), (loc_B, 'Dirty')): 'Suck',
             ((loc_B, 'Clean'), (loc_A, 'Dirty')): 'Suck',
             ((loc_B, 'Dirty'), (loc_B, 'Clean')): 'Left',
             ((loc_A, 'Dirty'), (loc_A, 'Clean'), (loc_B, 'Dirty')): 'Suck',
             ((loc_B, 'Dirty'), (loc_B, 'Clean'), (loc_A, 'Dirty')): 'Suck'}
    return Agent(TableDrivenAgentProgram(table))
```

How to write programs that produce rational behaviour from a reasonably short program than from an enormous table?

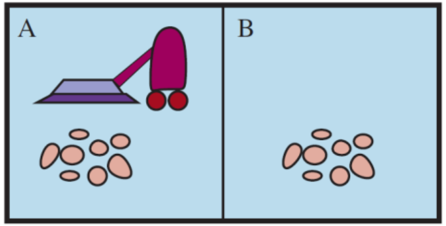
Agent Program : Reflex Agent

- Use the latest percept to determine the action
  - Simple and fast
  - Running away from a snake, moving hand away from a hot surface

  - Vacuum-world agent: $\mathbf{4}^t$ entries to **4** entries
  - Implemented using condition-action rules

function REFLEX-VACUUM-AGENT([*location*,*status*]) **returns** an action

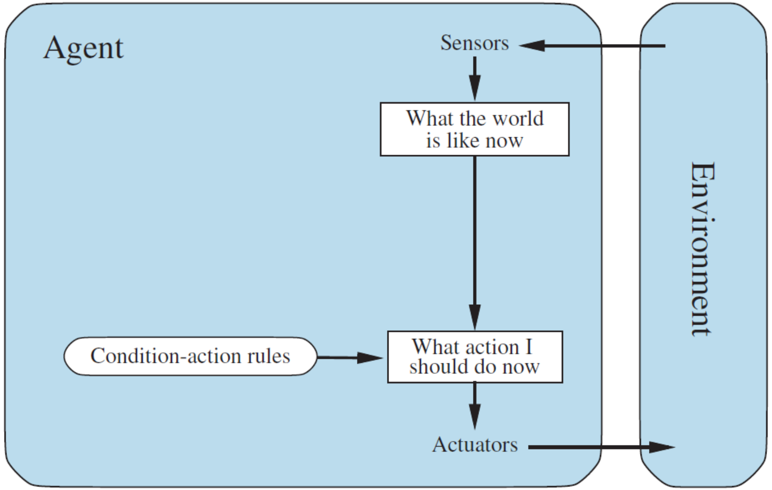
    if *status* = *Dirty* **then return** *Suck*
    else if *location* = *A* **then return** *Right*
    else if *location* = *B* **then return** *Left*



Agent Program : Reflex Agent

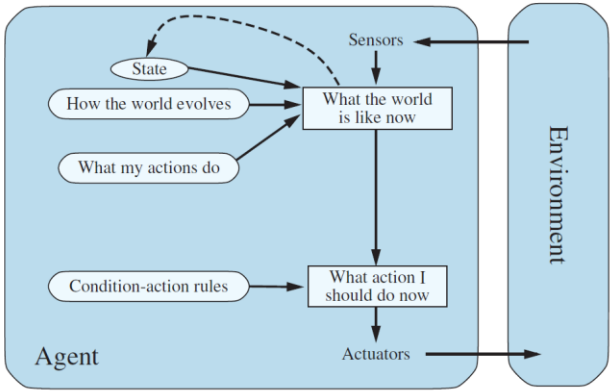
- Reflex agents are appropriate when the latest percept can correctly determine the rational action
  - Problematic in environments that are not fully observable
    - No location sensor in the vacuum-world can cause infinite loop
      - Randomization can help

  - How to deal with partial observability??



Model-Based Agent

- The agent maintains internal state that can be used to gain (an approximate) information about the unobserved aspects
- A model-based agent needs
  - A model of the evolution of the world and the result of agent's actions: **Transition Model**
    - Relationship between speed and distance covered
  - A model of how the state of the world is reflected in its percepts: **Sensor Model**
    - Darkness and on headlights means the sun has set



A model-based reflex agent.

Model-based Agent

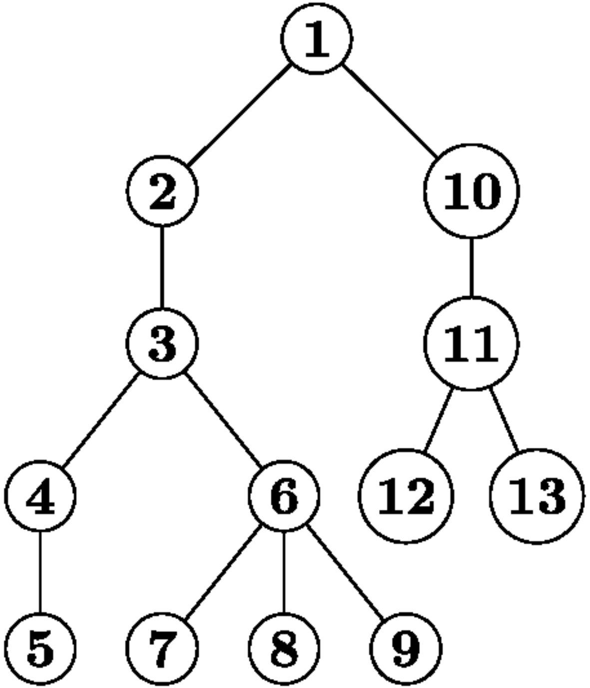
```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              transition_model, a description of how the next state depends on
                        the current state and action
              sensor_model, a description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

A Digression…

- DFS, BFS



Navigation example

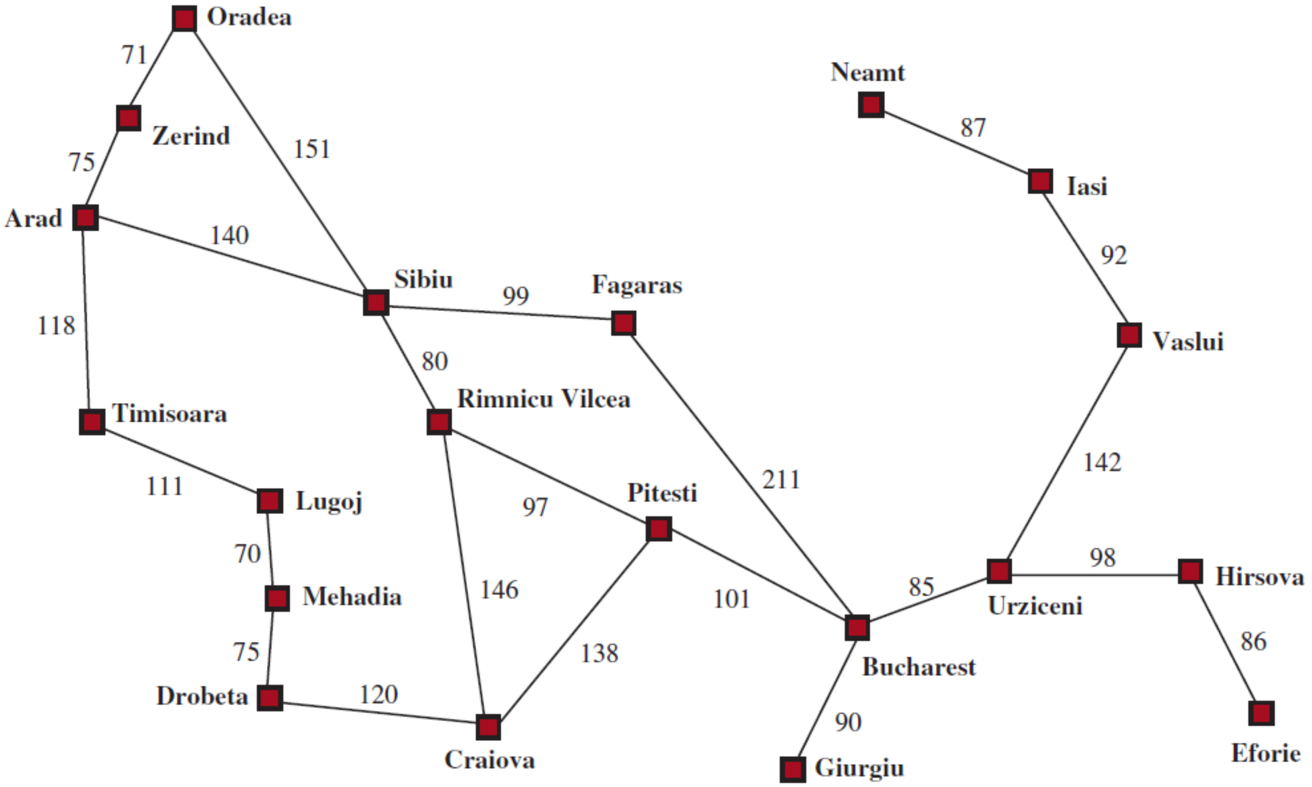


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Navigation example

- Find a *sequence* of actions that form a path to the destination (goal state)
  - Called problem-solving agent
  - The computational process it undertakes is called search

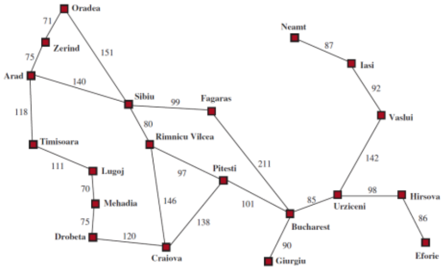
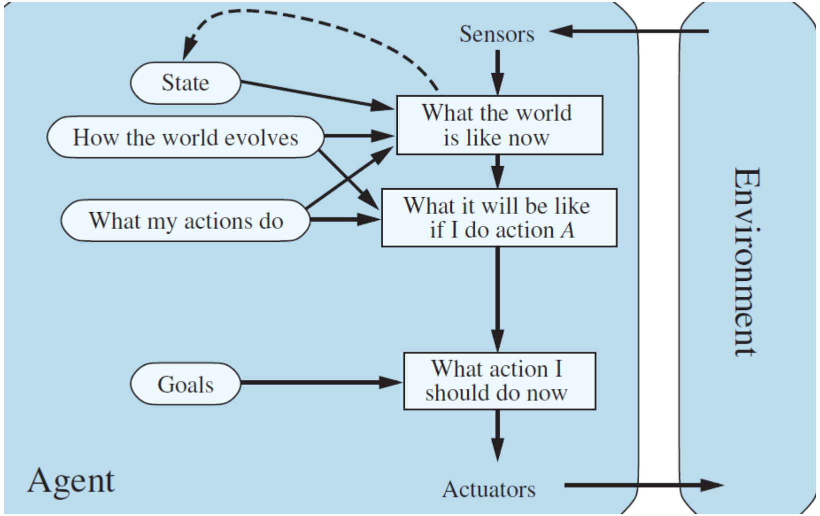


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

- Steps
  - Goal Formulation
  - Problem formulation: a description of the states and actions to reach the goal
- Search: simulates sequences of actions in its model
- Execution

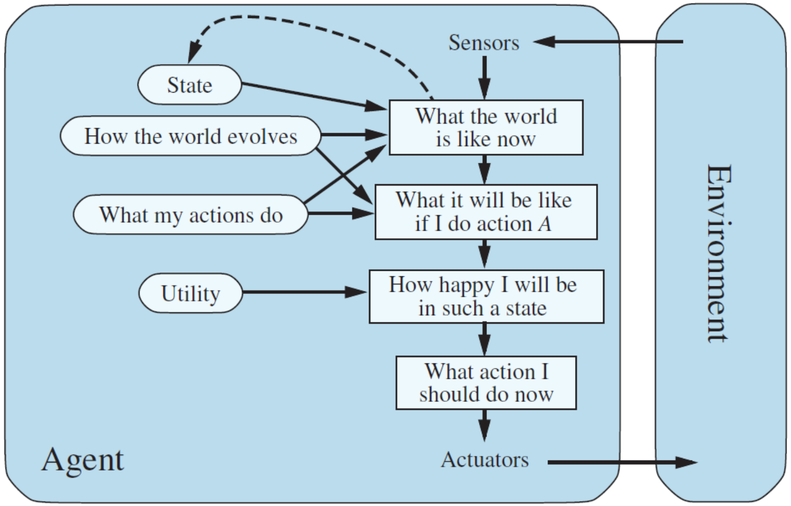
Goal-Based Agent

- When many seemingly equally rational actions but with potentially very different outcomes are available, knowledge of only the environment is not enough
  - What to do at an intersection?
- A goal is needed to select actions
  - No mapping from percept to actions
- Goal-based action selection:
  - Trivial if episodic
  - Search
  - Planning



- Goal-based agents are flexible – can change goals to do different tasks

Utility-Based Agents



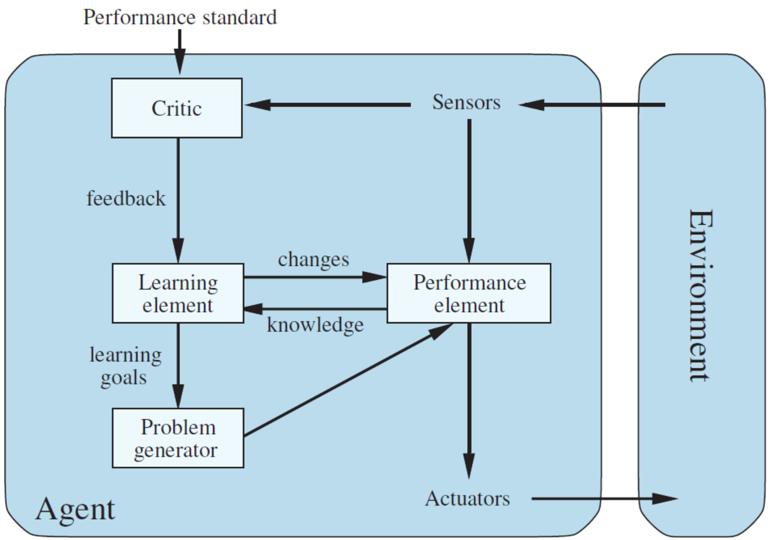
- Goals can be achieved in different ways
  - Goals provide binary classification
    - achieved (good) or not achieved (bad)
- Want to maximize the performance measure
- Utility is an internalization of performance measure provided aligned with the latter
  - Distinguishes between states
  - Allows evaluating tradeoffs
  - Allows factoring in likelihoods when many goals
    - Probability of sucess vs importance of goals

Utility-Based Agents

- In uncertain situations (environment, result of actions, sensor model etc.), one attempts to maximize **expected utility**
  - Probability and utility of each outcome
- Challenges
  - Utility-maximizing strategy
  - Model and track the environment
    - Perception, representation, reasoning and learning
  - Computational complexity makes attaining perfection difficult or even unachievable
- Transition model is not always necessary

Learning Agents

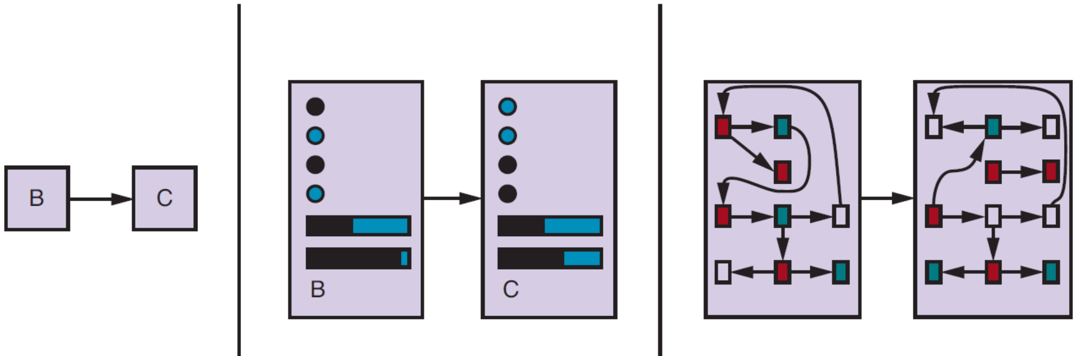
- Instead of writing (many, complex) agent programs, write a learning machine and teach it (Turing, 1950)
- Learning agents are needed in unknown environments
- Performance element selects actions
  - Was the entire agent before
- Learning element
  - makes improvements
  - Uses feedback from critic on its performance, modifies the performance element
- Problem generator suggests new actions for exploration
  - Can be sub-optimal in the short term



Agent representations

- **Atomic**: A state has no internal representation
  - Indivisible black box
- **Factored**: State is represented by a vector of values
- **Structured**: State includes objects and interacts with other objects

- From simplest to most complex
- From least to most expressive
- From least to most compact



(a) Atomic (b) Factored (c) Structured

- Reading: Chapter 2
- Assignments: PS 1, agent.ipynb, agent_problems.ipynb
- Project: Proposal due on August 21, before class, hard copy
- Next: Search, Chapter 3