INTELLIGENT AGENTS

Outline

- Introduction
- How Agents Should Act
- Structure of Intelligent Agents
- Environments

Based on the textbook by S. Russell & P. Norvig:

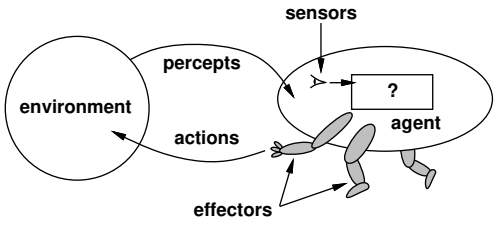
Artificial Intelligence, A Modern Approach, Chapter 2

© 2003 HUT / Laboratory for Theoretical Computer Science

INTRODUCTION

- Russell and Norvig define agents as follows:

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



© 2003 HUT / Laboratory for Theoretical Computer Science

Examples.

A physical robot

- Sensors: video camera, laser scanner, microphone, ...
- Effectors: motor, switch, display, speaker, ...

A software robot (softbot)

- Percepts: encoded bit strings
- Sensors and effectors:
  calls to operating system, libraries or other programs
- Calls to sensor programs provide input for the agent.

© 2003 HUT / Laboratory for Theoretical Computer Science

HOW AGENTS SHOULD ACT

- A *rational agent* should do the right thing, but *how* and *when* do we evaluate agent's success?
- A *performance measure* determines how successful an agent is (by an outside observer).

Problems:

- Self-deception: humans typically say they did not really want something after they are unsuccessful at getting it.
- Malpractice if performance is measured only instantly.
- You get what you ask for!
  (Performance measures have to be carefully chosen.)

© 2003 HUT / Laboratory for Theoretical Computer Science

Omniscience vs. Rationality

An omniscient agent

- ➤ knows the *actual* outcomes of its actions and acts accordingly.
- ➤ is impossible in reality.

Example. A person is crossing a street, as (s)he noticed a friend across the street and there is no traffic nearby.

- ➤ Is this person acting rationally if (s)he is crashed by a cargo door falling off a passing airplane?

☞ Rationality: expected success given what has been perceived.

© 2003 HUT / Laboratory for Theoretical Computer Science

Ideal Rational Agent

Rationality depends on four things:

- ➤ Performance measure which defines degree of success
- ➤ Percept sequence (complete perceptual history)
- ➤ Agent's knowledge about the environment
- ➤ Agent's actions

Definition. (*Ideal rational agent*)
For each possible percept sequence, an ideal rational agent should do whatever action which is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

© 2003 HUT / Laboratory for Theoretical Computer Science

Example. Often begin rational requires performing actions in order to acquire information about the environment.

- ➤ For instance, crossing a street without looking is too risky.

Example. A clock can be thought as a simple (even degenerate) agent that keeps moving its hands (or displaying digits) in the proper way.

- This can be thought as rational action given what kind of functionality one expects from a clock in general.
- However, many clocks are unable to take changing time zones into account automatically. This is quite acceptable if the clock does not have a mechanism for perceiving time zones.

© 2003 HUT / Laboratory for Theoretical Computer Science

Mapping Percept Sequences to Actions

- ➤ Agent's behavior depends only on its percept sequence to date.
- ➤ An ideal agent can be designed by

  "specifying which action an agent ought to take in response to any given percept sequence".

- ➤ The mapping can be represented as a table or as a program.

Example. Consider a calculator agent that computes square-roots of positive integers (accurate to 15 decimals).

- ➤ Approach 1: store square roots in a very large table.
- ➤ Approach 2: implement the ideal mapping as a program.

The latter approach is clearly more compact and flexible.

© 2003 HUT / Laboratory for Theoretical Computer Science

Autonomy

- An agent lacks autonomy if its actions depend solely on its built-in knowledge about the environment.
- A system is *autonomous* to the extent that its behavior is determined by its own experience.
- Flexible operation in a variety of environments demands ability to learn (in addition to initial knowledge).

Example. After digging its nest and laying its eggs, a dung beetle fetches a ball of dung to plug the entrance.

- Even if the ball is removed from its grasp en route, the beetle continues and mimics the procedure to the very end.

© 2003 HUT / Laboratory for Theoretical Computer Science

STRUCTURE OF INTELLIGENT AGENTS

- The goal is to design an *agent program* which implements the mapping from percepts to actions.
- An *architecture* is a computing device that makes percepts available, runs the program and feeds action choices to effectors.
- Summarizing: **agent = architecture + program**.
- Agents can be roughly categorized by identifying their *percepts*, *actions*, *goals* and *environments* (so-called PAGE descriptions).

© 2003 HUT / Laboratory for Theoretical Computer Science

Examples. PAGE descriptions for some types of agents.

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize student's score on test	Set of students

© 2003 HUT / Laboratory for Theoretical Computer Science

Agent Programs

A skeleton for agent programs:

- A single percept is obtained as input.
- Memory is used for storing the percept history (if necessary).
- The program chooses and outputs an action to be executed next.

```
function SKELETON-AGENT( percept) returns action
  static: memory, the agent's memory of the world

  memory ← UPDATE-MEMORY(memory, percept)
  action ← CHOOSE-BEST-ACTION(memory)
  memory ← UPDATE-MEMORY(memory, action)
  return action
```

- The performance measure is not a part of the program.

© 2003 HUT / Laboratory for Theoretical Computer Science

Using Lookup Tables

- An agent program that looks up the action from a table:

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

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

- Drawbacks of lookup table agents:
  1. The lookup table becomes easily very large (a chess playing agent would need a table with 35^{100} entries).
  2. The table is difficult to build and maintain.
  3. The resulting agent does not have autonomy at all.
  4. It would take forever to learn the right values for all entries.

© 2003 HUT / Laboratory for Theoretical Computer Science

Example

- Consider designing an automated taxi driving agent with the following PAGE description:

Agent Type	Percepts	Actions	Goals	Environment
Taxi driver	Cameras, speedometer, GPS, sonar, microphone	Steer, accelerate, brake, talk to passenger	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers

- The full driving task is extremely *open-ended*: an unlimited number of novel combinations of circumstances can arise.
- Performance measures: (i) getting to the correct destination, (ii) minimizing fuel consumption, trip time/cost, and traffic violations, (iii) maximizing safety, passenger comfort, and profits.

© 2003 HUT / Laboratory for Theoretical Computer Science

Different Kinds of Agent Programs

In the sequel, we will consider four kinds of agent programs:

- Simple reflex agent
- Agents that keep track of the world
- Goal-based agents
- Utility-based agents

© 2003 HUT / Laboratory for Theoretical Computer Science

Simple Reflex Agents

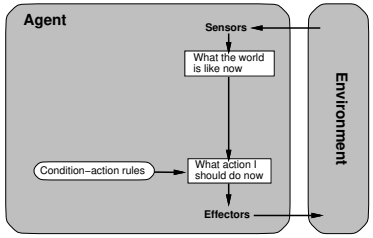
- *Condition-action rules* provide a way to represent common regularities appearing in input/output associations:

  if car-in-front-is-braking **then** initialize-braking

- It is even possible to learn such rules on the fly.
- Humans also have many such connections some of which are *learned responses* and some of which are *innate reflexes* (such as eye blinking in order to protect the eye).
- Mimicking reflexes of living creatures, a **reflex agent** chooses the next action on the basis of the current percept.
- No track of the world/environment is kept.

© 2003 HUT / Laboratory for Theoretical Computer Science

- The structure of a simple reflex agent as a schematic diagram and the corresponding skeletal agent program:




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

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

- Rules provide an efficient representation, but one problem is that decision making is seldom possible on the basis of a single percept.

© 2003 HUT / Laboratory for Theoretical Computer Science

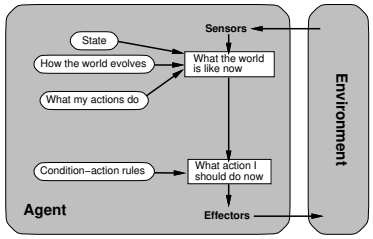
Reflex Agents with Internal State

- The choice of actions may depend on the entire percept history.
- Sensors do not necessarily provide access to the complete state of the environment.
- The agent keeps track of the world by extracting relevant information from percepts and storing it in its memory.
- Using a model of the environment, the agent may try to estimate
  1. how the environment evolves in the (near) future, and
  2. how the environment is affected by the agent's actions.

Example. In our taxi driving example, actions may depend on the state, e.g. the position of an overtaking car.

© 2003 HUT / Laboratory for Theoretical Computer Science

- A schematic diagram and a skeletal agent program for a reflex agent with an internal state:




```
function REFLEX-AGENT-WITH-STATE( percept) returns action
  static: state, a description of the current world state
          rules, a set of condition-action rules

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

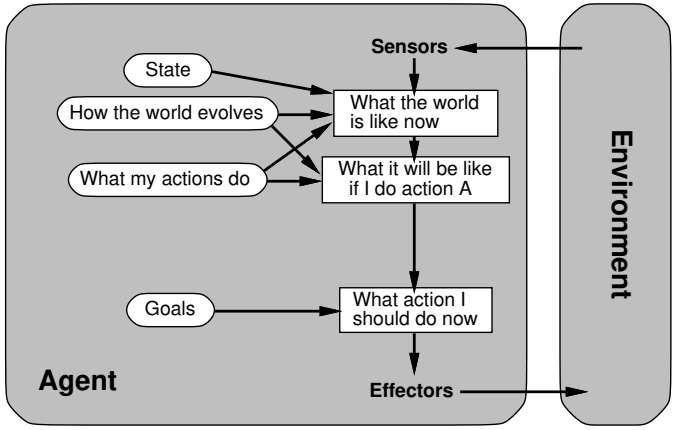
© 2003 HUT / Laboratory for Theoretical Computer Science

Goal-based agents

- Knowing about the current state of the environment is not necessarily enough for deciding what to do.
- In addition, the agent may need **goals** to distinguish which situations are desirable and which are not.
- Goal information can be combined with the agent's knowledge about the results of possible actions in order to choose an action leading to a goal.
- Problem: goals are not necessarily achievable by a single action.
- **Search** and **planning** are subfields of AI devoted to finding actions sequences that achieve the agent's goals.

© 2003 HUT / Laboratory for Theoretical Computer Science

- A schematic diagram for a goal-based agent:



- Additional flexibility compared to previous designs: the behavior of a goal-based agent can be changed by changing its goal(s).

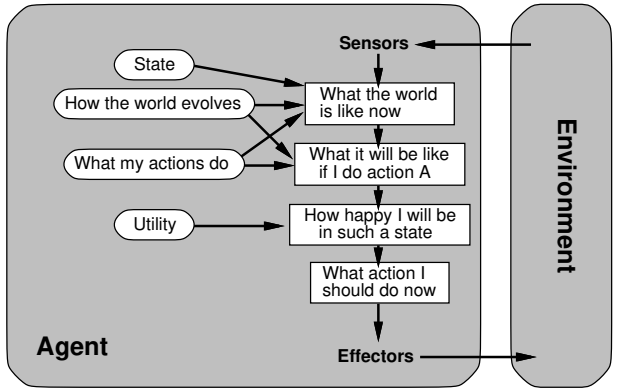
© 2003 HUT / Laboratory for Theoretical Computer Science

Utility-based agents

- Goals alone are not sufficient for decision making if there are several ways of achieving them.
- Further problem: agents may have several conflicting goals that cannot be achieved simultaneously.
- If an agent prefers one world state to another state then the former state has higher utility for the agent.
- Utility is a function that maps a state onto a real number.
- Utility can be used for (i) choosing the best plan, (ii) resolving conflicts among goals, and (iii) estimating the successfulness of an agent if the outcomes of actions are uncertain.

© 2003 HUT / Laboratory for Theoretical Computer Science

- A schematic diagram for a utility-based agent:



- An agent that possesses an *explicit* utility function can make rational decisions, but may have to compare the utilities achieved by different courses of actions.

© 2003 HUT / Laboratory for Theoretical Computer Science

ENVIRONMENTS

- All the preceding agent designs are based on the same interconnection between an agent and its environment:
  1. The agent performs actions on the environment.
  2. The environment provides percepts to the agent.
- Environments can be distinguished by their properties which in turn affect the design of respective agents.
- Environment programs (that simulate particular environments) can be used as testbeds for agent programs.

© 2003 HUT / Laboratory for Theoretical Computer Science

Properties of Environments

Environments can be categorized by several aspects such as

- *Accessible vs. inaccessible* state of the environment
  Also: *effectively accessible* (w.r.t. choice of actions)
- *Deterministic vs. nondeterministic* outcomes of agent's actions
- *Episodic vs. nonepisodic*
- *Static vs. dynamic*
  Also: *semidynamic* (perfomance degrades over time)
- *Discrete vs. continuous*

© 2003 HUT / Laboratory for Theoretical Computer Science

Examples. Analyzing properties of a number of familiar environments.

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

- Some of the properties are dependent on how the environments and agents are conceptualized.

© 2003 HUT / Laboratory for Theoretical Computer Science

Programs Simulating Environments

```
procedure RUN-ENVIRONMENT(state, UPDATE-FN, agents, termination)
  inputs: state, the initial state of the environment
          UPDATE-FN, function to modify the environment
          agents, a set of agents
          termination, a predicate to test when we are done

  repeat
     for each agent in agents do
        PERCEPT[agent] ← GET-PERCEPT(agent, state)
     end
     for each agent in agents do
        ACTION[agent] ← PROGRAM[agent](PERCEPT[agent])
     end
     state ← UPDATE-FN(actions, agents, state)
  until termination(state)
```

- Agents are typically designed to work correctly in a class of environments (that has to be covered by a simulator somehow).
- Agent programs should not have other access than percepts to the state of the program simulating their environment!

© 2003 HUT / Laboratory for Theoretical Computer Science

- The performance of agents can be measured by inserting special measurement code to simulator programs.

```
function RUN-EVAL-ENVIRONMENT(state, UPDATE-FN, agents,
                              termination, PERFORMANCE-FN) returns scores
  local variables: scores, a vector the same size as agents, all 0

  repeat
     for each agent in agents do
        PERCEPT[agent] ← GET-PERCEPT(agent, state)
     end
     for each agent in agents do
        ACTION[agent] ← PROGRAM[agent](PERCEPT[agent])
     end
     state ← UPDATE-FN(actions, agents, state)
     scores ← PERFORMANCE-FN(scores, agents, state)
  until termination(state)
  return scores                                          /* change */
```

© 2003 HUT / Laboratory for Theoretical Computer Science

SUMMARY

- An agent program maps a percept (sequence) to an action.
- Agent = architecture + agent program
- An ideal agent maximizes its performance measure.
- An agent can be described by its percepts, actions, goals and environment (PAGE description).
- Various agent types: reflex agents with(out) internal state, goal-based agents, utility-based agents
- Important aspects of agent program design: efficiency, compactness, flexibility

© 2003 HUT / Laboratory for Theoretical Computer Science

(AI) TECHNIQUES FOR AGENT DESIGN

- Knowledge representation and reasoning
- Search algorithms
- Planning
- Learning
- Decision theory
- Natural language processing
- Perception
- Robotics

© 2003 HUT / Laboratory for Theoretical Computer Science

Knowledge-based Agents

```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

- Incomplete information
- Uncertainty
- Probability

© 2003 HUT / Laboratory for Theoretical Computer Science

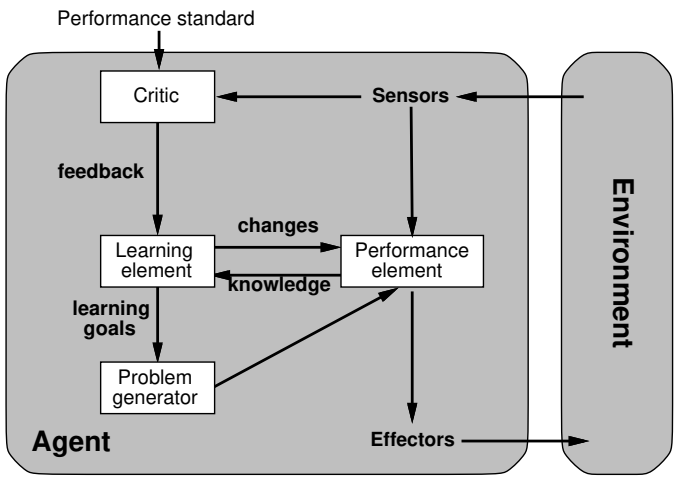
Planning Agents

```
function SIMPLE-PLANNING-AGENT(percept) returns an action
  static: KB, a knowledge base (includes action descriptions)
          p, a plan, initially NoPlan
          t, a counter, initially 0, indicating time
  local variables: G, a goal
                   current, a current state description

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  current ← STATE-DESCRIPTION(KB, t)
  if p = NoPlan then
     G ← ASK(KB, MAKE-GOAL-QUERY(t))
     p ← IDEAL-PLANNER(current, G, KB)
  if p = NoPlan or p is empty then action ← NoOp
  else
     action ← FIRST(p)
     p ← REST(p)
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

© 2003 HUT / Laboratory for Theoretical Computer Science

Learning Agents



© 2003 HUT / Laboratory for Theoretical Computer Science

QUESTIONS

- Analyze soccer playing agents by writing down
  1. a PAGE description, and
  2. main properties of the environment.
- Consider the following designs for soccer playing agents:
  1. Simple reflex agent
  2. Reflex agent with internal state
  3. Agent with explicit goals
  4. Utility-based agent
  5. Planning agent
  6. Learning agent

  What kind of functionality can be implemented using these?

© 2003 HUT / Laboratory for Theoretical Computer Science