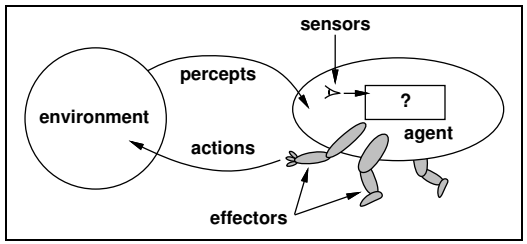
Intelligent Agents

Outline

- What is an agent?
- How agents should act
- Structure of agents
- Environments
- (AI) techniques for agents

What is an Agent?

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



How Agents Should Act

- A rational agent: does the right thing.
- Performance measure
  (how successful an agent is; by an outside observer)
- omniscience vs. rationality
  Rationality: expected success given what has been perceived.

- Percept sequence (complete perceptual history)
- Agent's knowledge about the environment
- Agent's actions

Ideal rational agent:
For each possible percept sequence, an ideal rational agent should do whatever action 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.

Mapping from percept sequences to actions

- Agent's behavior depends only on its percept sequence
- An agent can be described by
  specifying which action an agent ought to take in
  respond to any given percept sequence.
- Mapping vs. agent design

Autonomy

- A system is autonomous to the extent that its behavior is determined by its own experience.
- built-in knowledge + learning
- flexibility
- operation in a variety of environments

Structure of Intelligent Agents

- Agent program: implements mapping from percept sequences to actions
- Architecture: makes percepts available, runs the program and feeds action choices to effectors

  agent = architecture + program

- Agent types:
  Percepts, actions, goals, environment (PAGE)

PAGE descriptions

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

Agent Programs

- Single percept as input
- Memory for percept sequence
- Performance measure is not part of the program

```
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
```


Lookup table agent

```
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
```

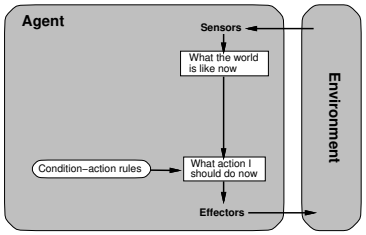
- Table very large
- Table difficult to build and update
- Autonomy lacking

Example: taxi driver agent

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

Simple reflex agent: condition-action rules for current percept

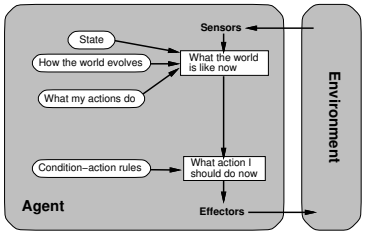
if car-in-front-is-braking **then** initialize-braking




```
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
```

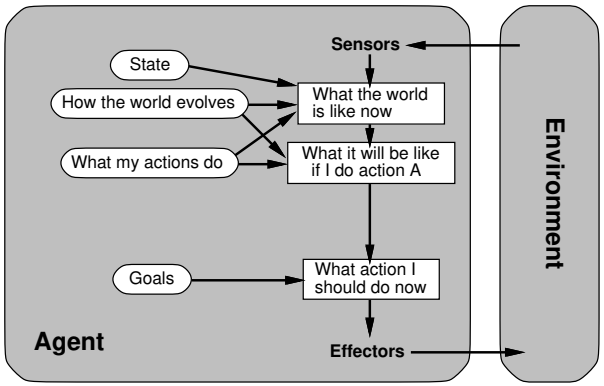
Agents that keep track of the world: internal state



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*

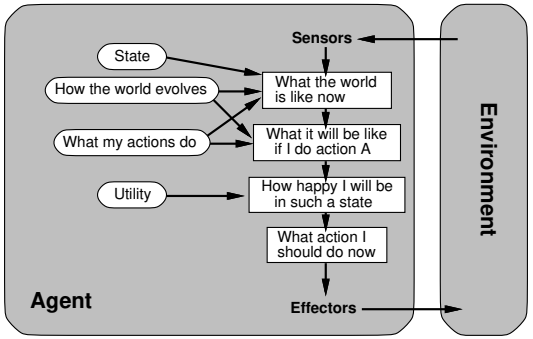
Goal-based agents



Utility-based agents

Utility: maps a state onto a real number

Conflicting goals, uncertainty about achieving goals



Environments

- Accessible vs. inaccesible
- Deterministic vs. nondeterministic
- Episodic vs. nonepisodic
- Static vs. dynamic
- Discrete vs. continuous

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

Simulating the environment

```
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)
```



```
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 */
```


(AI) techniques for agents

- knowledge representation and reasoning
- search
- planning
- learning
- decision theory
- natural language processing
- perception, robotics

Knowledge-based agent

```
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
```

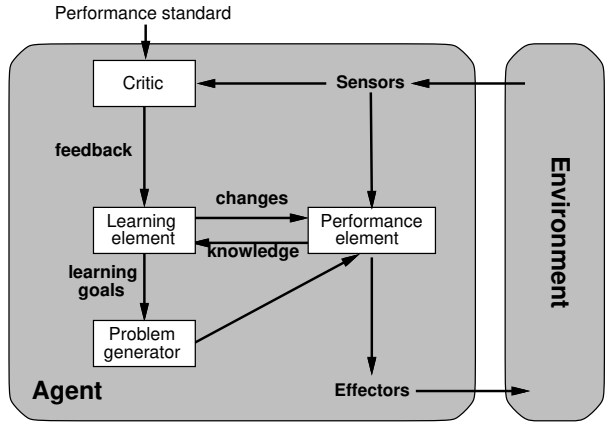
- Incomplete information
- Uncertainty
- Probability

Planning agent

```
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
```


Learning agent



Summary

- agent = architecture + agent program
- ideal agent: maximizes its performance measure
- agent program: maps percept to action
- agent program design: efficiency, compactness, flexibility
- percepts, actions, goals, environment
- reflex agents, goal-based agents, utility-based agents