Intelligent Agent

1

Conceptual Design of an Intelligent Agent

Perception

Reasoning

Action

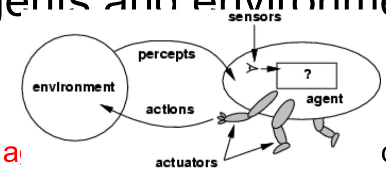
2

Agents

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators
- Human agent: eyes, ears, and other organs for sensors; hands, legs, mouth, and other body parts for actuators
- Robotic agent: cameras and infrared range finders for sensors; various motors for actuators

3

Agents and environments



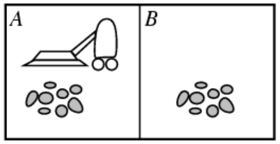
- The a cept histories to actions:

  [f: $\mathcal{P}^* \rightarrow \mathcal{A}$]

- The agent program runs on the physical architecture to produce f
- agent = architecture + program

4

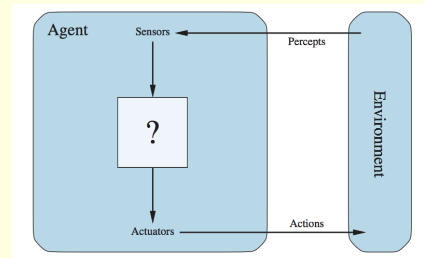
Vacuum-cleaner world



- Percepts: location and contents, e.g., [A,Dirty]
- Actions: *Left*, *Right*, *Suck*, *NoOp*

5

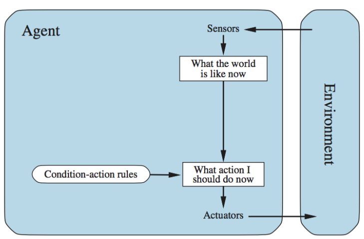
Conceptual Design



An agent perceives its environment with sensors and acts on the environment using actuators.

6

REFLEX AGENT



An agent perceives its environment with sensors and acts on the environment using actuators.

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

7

Rational agents

- An agent should strive to "do the right thing", based on what it can perceive and the actions it can perform. The right action is the one that will cause the agent to be most successful
- Performance measure: An objective criterion for success of an agent's behavior
- E.g., performance measure of a vacuum-cleaner agent could be amount of dirt cleaned up, amount of time taken, amount of electricity consumed, amount of noise generated, etc.

8

Rational agents

- Rational Agent: For each possible percept sequence, a rational agent should
  - select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

9

Rational agents

- Rationality is distinct from omniscience (all-knowing with infinite knowledge)
- Agents can perform actions in order to modify future percepts so as to obtain useful information (information gathering, exploration)
- An agent is autonomous if its behavior is determined by its own experience (with ability to learn and adapt)

10

PEAS

- PEAS: Performance, Environment, Actuators, Sensors
- Must first specify the setting for intelligent agent design
- Consider, e.g., the task of designing an automated taxi driver:
  - Performance measure
    - Safe, fast, legal, comfortable trip, maximize profit
  - Environment
    - Roads, other traffics, pedestrians, customers
  - Actuators
    - Steering, accelerator, brake, signal, horn, display
  - Sensors
    - Cameras, sonar, GPS, odometer, keyboard etc..

11

PEAS

- Agent: Medical diagnosis system
  - Performance measure: Healthy patient, minimize costs, lawsuits
  - Environment: Patient, hospital, staff
  - Actuators: Screen display (questions, tests, diagnoses, treatments, referrals)
  - Sensors: Keyboard (entry of symptoms, findings, patient's answers)

12

PEAS

- Agent: Part-picking robot
  - Performance measure: Percentage of parts in correct bins
  - Environment: Conveyor belt with parts, bins
  - Actuators: Jointed arm and hand
  - Sensors: Camera, joint angle sensors

13

PEAS

- Agent: Interactive English tutor
  - Performance measure: Maximize student's score on test
  - Environment: Set of students
  - Actuators: Screen display (exercises, suggestions, corrections)
  - Sensors: Keyboard

14

Environment types

- Fully observable (vs. partially observable): An agent's sensors give it access to the complete state of the environment at each point in time.
- Deterministic (vs. stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent. (If the environment is deterministic except for the actions of other agents, then the environment is strategic)
- Episodic (vs. sequential): The agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself

15

Environment types

- Static (vs. dynamic): The environment is unchanged while an agent is deliberating. (The environment is semidynamic if the environment itself does not change with the passage of time but the agent's performance score does)
- Discrete (vs. continuous): A limited number of distinct, clearly defined percepts and actions.
- Single agent (vs. multiagent): An agent operating by itself in an environment.

16

Environment types

	Chess with a clock	Chess without a clock	Taxi driving
Fully observable	Yes	Yes	No
Deterministic	Strategic	Strategic	No
Episodic	No	No	No
Static	Semi	Yes	No
Discrete	Yes	Yes	No
Single agent	No	No	No

- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic,

  sequential, dynamic, continuous, multi-agent

17

Agent functions and programs

- An agent is completely specified by the agent function mapping percept sequences to actions
- One agent function (or a small equivalence class) is rational
- Aim: find a way to implement the rational agent function concisely

18

Table-lookup agent

Persistent: *percepts* (a sequence initially empty)
table (table of actions initially fully specified)
Append *percept* to the end of percepts
Action <- LookUp(*percepts,table*)
Return *action*

- Drawbacks:
  - Huge table
  - Take a long time to build the table
  - No autonomy
  - Even with learning, need a long time to learn the table entries

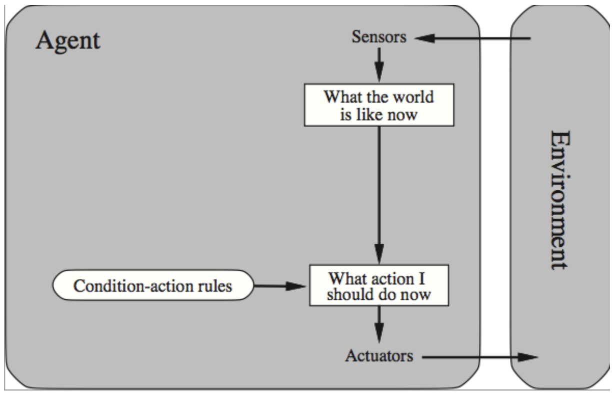
19

Agent types

- Four basic types in order of increasing generality:
- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

20

Simple reflex agents



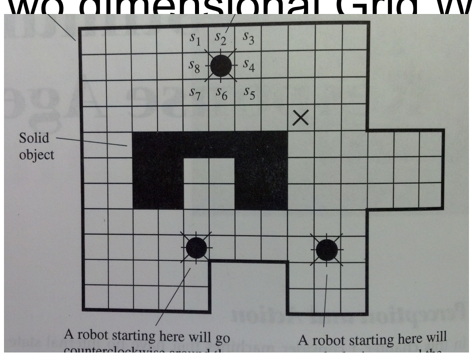
21

Simple reflex agents

Function Simple-Reflex-Agent(*percept*) return action
persistent: *rules* (a set of condition action rules)
state <- Interpret-Input(*percept*)
rule <- Rule-Match(*state,rules*)
action<- *rule*.action
Return *action*

22

Two dimensional Grid World

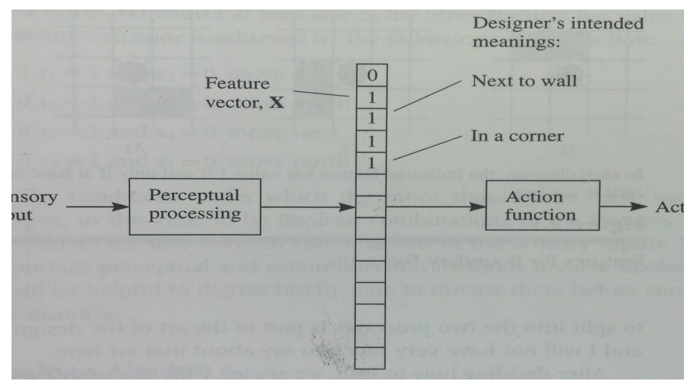


If the robot were in position marked by X the values of the sensory inputs (starting from s1) would be (0,0,0,0,0,0,1,0)

(Artificial Intelligence: A new synthesis by Nils J. Nilsson)

23

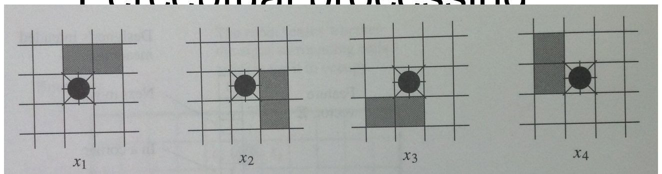
Percept Action Components



(Artificial Intelligence: A new synthesis by Nils J. Nilsson)

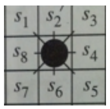
24

Perceptual processing



In each diagram, the indicated feature has value 1 if and only if at least one of the shaded cells is *not* free.

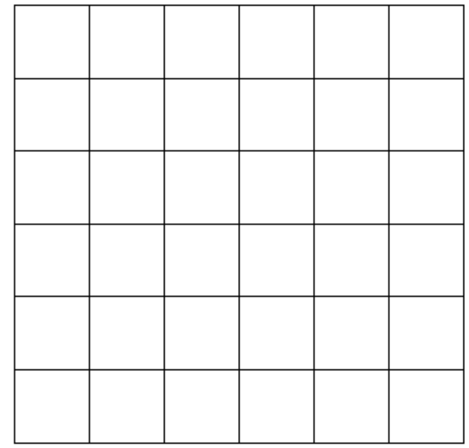
X1=1 iff if s2=1 or s3=1
X2=1 iff if s4=1 or s5=1
X3=1 iff if s6=1 or s7=1
X4=1 iff if s8=1 or s1=1



if $x_1 = 1$ and $x_2 = 0$, move east
if $x_2 = 1$ and $x_3 = 0$, move south
if $x_3 = 1$ and $x_4 = 0$, move west
if $x_4 = 1$ and $x_1 = 0$, move north

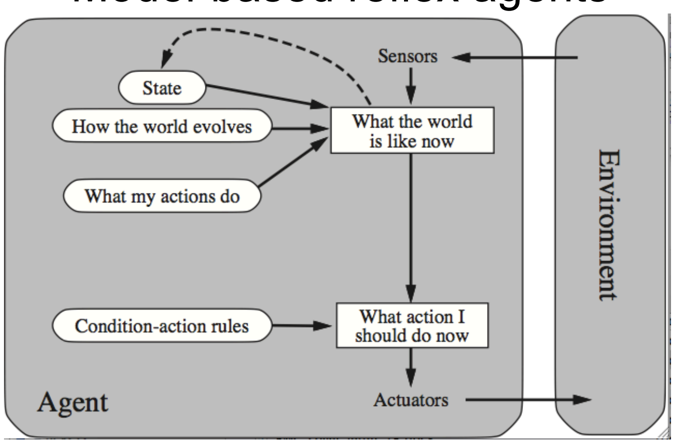
(Artificial Intelligence: A new synthesis by Nils J. Nilsson)

25



26

Model-based reflex agents



27

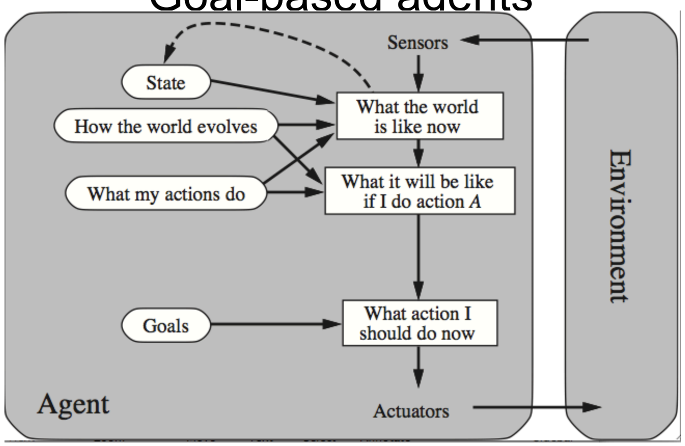
Model-based reflex agents

function MODEL-BASED-REFLEX-AGENT(*percept*) returns an action
persistent *state*, the agent's current conception of the world state
model, a *description* of how the next state depends on current state and action
rules, a set of condition—action rules
action, the most recent action, initially none

state UPDATE-STATE(*state*, *action*, *percept*, *model*)
rule. ← RULE MATCH(*state*,
action *rule*.ACTION
return *action*

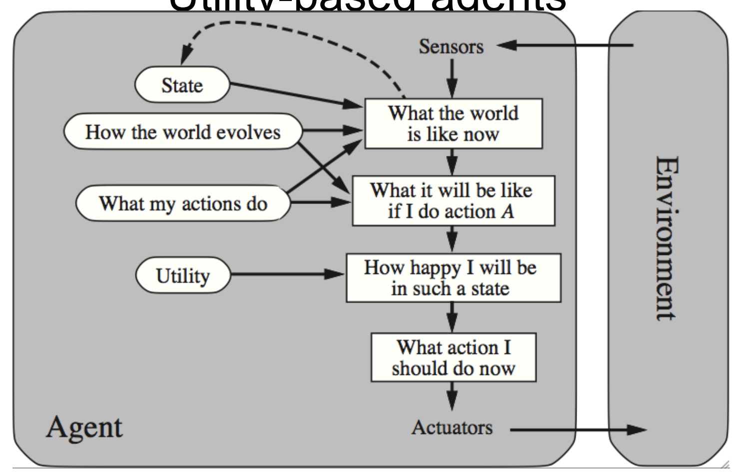
28

Goal-based agents



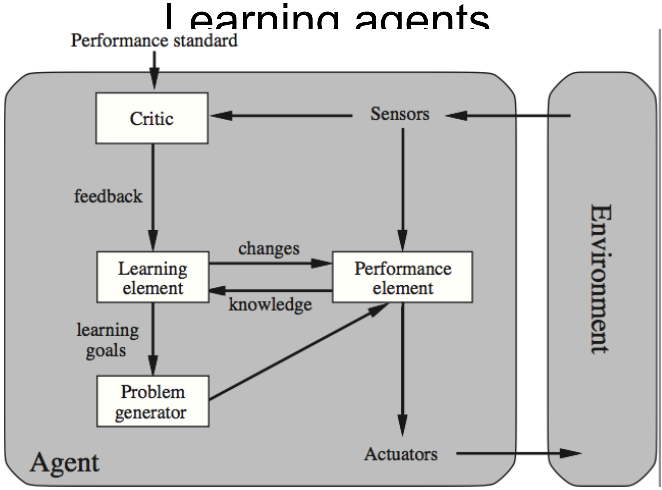
29

Utility-based agents



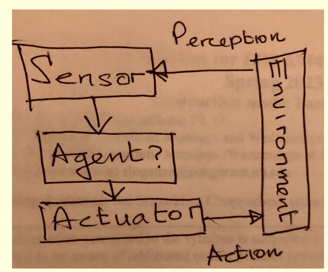
30

Learning agents



31

Frame of AI



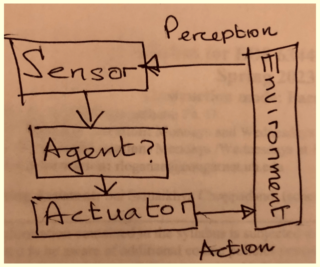
An agent perceives its environment with sensors and acts on environment using actuators

Car:

sensors

32

Frame of AI



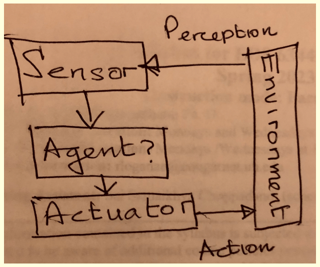
An agent perceives its environment with sensors and acts on environment using actuators

Car:

sensors: Camera, lidar, speed gauge
Actuators:

33

Frame of AI



An agent perceives its environment with sensors and acts on environment using actuators

Car:

sensors: Camera, lidar, speed gauge
Actuators: gas petal, steering wheel, brake petal

34

Rationality

A rational agent chooses actions that maximize expected utility.

35

Rationality

A rational agent chooses actions that maximize expected utility.

Assume that an agent has a goal and a cost

The agent must reach the goal with the lowest cost.

36

Agent Design

The environment largely determines the agent design.

Fully/partially observability -> agent capability memory

37

Logical Agent

function KB-AGENT(percept) **returns** an action
persistent: 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

A generic Knowledge-based agent.
Given a percept, the agent adds the percept to its knowledge base, ask the knowledge base for the best action, and tells the knowledge base that it has in fact taken the action

38

Propositional Logic

$$
\begin{array}{rcl}
\mathit{Sentence} & \to & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \to & \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} & \to & (\ \mathit{Sentence}\ ) \\
& \mid & \neg\ \mathit{Sentence} \\
& \mid & \mathit{Sentence} \land \mathit{Sentence} \\
& \mid & \mathit{Sentence} \lor \mathit{Sentence} \\
& \mid & \mathit{Sentence} \Rightarrow \mathit{Sentence} \\
& \mid & \mathit{Sentence} \Leftrightarrow \mathit{Sentence} \\
\\
\text{OPERATOR PRECEDENCE} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

P	Q	$\neg P$	$P \land Q$	$P \lor Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

39

Propositional Logic

$$
\begin{array}{rcll}
(\alpha \land \beta) & \equiv & (\beta \land \alpha) & \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv & (\beta \lor \alpha) & \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv & (\alpha \land (\beta \land \gamma)) & \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv & (\alpha \lor (\beta \lor \gamma)) & \text{associativity of } \lor \\
\neg(\neg\alpha) & \equiv & \alpha & \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\beta \Rightarrow \neg\alpha) & \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\alpha \lor \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv & (\neg\alpha \lor \neg\beta) & \text{De Morgan} \\
\neg(\alpha \lor \beta) & \equiv & (\neg\alpha \land \neg\beta) & \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv & ((\alpha \land \beta) \lor (\alpha \land \gamma)) & \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv & ((\alpha \lor \beta) \land (\alpha \lor \gamma)) & \text{distributivity of } \lor \text{ over } \land
\end{array}
$$

40

Propositional Logic

```
function PL-RESOLUTION(KB, α) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
          α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
  new ← { }
  while true do
     for each pair of clauses Ci, Cj in clauses do
        resolvents ← PL-RESOLVE(Ci, Cj)
        if resolvents contains the empty clause then return true
        new ← new ∪ resolvents
     if new ⊆ clauses then return false
     clauses ← clauses ∪ new
```

41

Propositional Logic

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is initially the number of symbols in clause c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  queue ← a queue of symbols, initially symbols known to be true in KB

  while queue is not empty do
     p ← POP(queue)
     if p = q then return true
     if inferred[p] = false then
        inferred[p] ← true
        for each clause c in KB where p is in c.PREMISE do
           decrement count[c]
           if count[c] = 0 then add c.CONCLUSION to queue
  return false
```

42

Propositional Logic

```
function DPLL-SATISFIABLE?(s) returns true or false
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(clauses, symbols, { })

function DPLL(clauses, symbols, model) returns true or false

  if every clause in clauses is true in model then return true
  if some clause in clauses is false in model then return false
  P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P, value ← FIND-UNIT-CLAUSE(clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P ← FIRST(symbols); rest ← REST(symbols)
  return DPLL(clauses, rest, model ∪ {P=true}) or
         DPLL(clauses, rest, model ∪ {P=false}))
```

43

Propositional Logic

```
function WALKSAT(clauses, p, max_flips) returns a satisfying model or failure
  inputs: clauses, a set of clauses in propositional logic
          p, the probability of choosing to do a "random walk" move, typically around 0.5
          max_flips, number of value flips allowed before giving up

  model ← a random assignment of true/false to the symbols in clauses
  for each i = 1 to max_flips do
     if model satisfies clauses then return model
     clause ← a randomly selected clause from clauses that is false in model
     if RANDOM(0, 1) ≤ p then
        flip the value in model of a randomly selected symbol from clause
     else flip whichever symbol in clause maximizes the number of satisfied clauses
  return failure
```

Figure 7.18 The WALKSAT algorithm for checking satisfiability by randomly flipping the values of variables. Many versions of the algorithm exist.

44

Satisfiability Problem

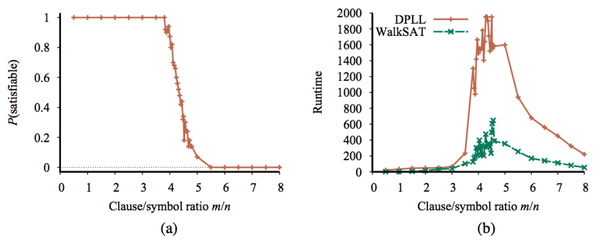


Figure 7.19 (a) Graph showing the probability that a random 3-CNF sentence with $n = 50$ symbols is satisfiable, as a function of the clause/symbol ratio m/n. (b) Graph of the median run time (measured in number of iterations) for both DPLL and WALKSAT on random 3-CNF sentences. The most difficult problems have a clause/symbol ratio of about 4.3.

45

46