Unit-1

Concept of AI:

We know intelligence is so important to us. We tried to understand how we think, how a mere handful of matter can perceive, understand, predict and manipulate a world far larger and complicated than itself. The field of AI goes further attempts not just to understand but also to build intelligent entities.

AI encompasses a huge variety of subfields ranging from the general (learning and perception) to the specific such as playing chess, proving mathematical theorems, writing poetry, driving a car on a crowded street.

AI definitions concerned with thought processes and reasoning.

Thinking Humanly: The exciting new effort to make computers think machines with minds in the full and literal sense(Haugeland 1985)

The automation of activities that we associate human thinking, activities such as decision making, problem solving, learning(Bellman 1978)

Thinking Rationally: The study of ability to think clearly through the use of computational models. (charniak and Mc Dermott 1985)

The study of computations that make it possible to perceive, reason and act. (Winston 1992)

Acting Humanly: The art of creating machines that perform functions that require intelligence when performed by people (Kurzweil 1990)

The study of how to make computers do things at which, at the moment people do better. (knight 1991)

Acting Rationally:

Computational Intelligence is study of the design of intelligent agents(poole 1998)

AI is concerned about intelligent behavior in artifacts(Nilsson 1998)

thinking Humanly:

Turing test proposed by Alan turing to provide satisfactory operational definition of intelligence

A computer passes the test if a human interrogator after posing some written questions cannot tell whether the written response came from a person or from a computer.

The computer would need following capabilities:

Natural language processing, Knowledge representation, automated reasoning, machine learning, computer vision and robotics.

Thinking Humanly: Cognitive modelling approach

If we are going to say that given program thinks like human , we must have some way of determining how humans think. If the programs input and output behavior matches

corresponding human behavior that is the evidence of the program's mechanism could also be operating in humans.

Thinking rationally: The laws of thought approach

The greek Philosopher Aristotle was one of the first attempt to codify right thinking . His syllogisms provided patterns for argument structures that yield correct conclusions when given correct premises

The two main obstacles of this approach are

Informal knowledge and it in formal form

Second solving a problem in principle is different from solving in practice

Acting Rationally:

An agent that acts something, that operate autonomously, persist over a prolonged time period, adapt to change

A rational agent is one acts so as to achieve the best outcome when there is uncertainty the best expected outcome.

All the skills needed for the turing test also needed by an agent to act rationally.

History of AI:

Aristotle (384 -322AD) was the first to formulate a precise set of law govering the rational part of the mind. He developed an informal system of syllogisms for proper reasoning which in principle allowed one to generate conclusions given initial premises.

George Boole (1815 -1864)worked on propositional or Boolean logic. In 1879, Gottlob extended Boolean logic to include objects and relations, creating first ordered logic. Alfred Tarski (1902 -1983) introduced a theory of reference that show how to relate the logic to objects in the real world.

A Persian mathematician, al-Khowarazmi introduced the algorithms in 9^{th} century. In 1930 Kurt Godel showed that there exist an effective procedure to prove any true statement in the first order logic.

Alan Turing (1912- 1954) characterize exactly which functions are computable capable of being computed. His thesis states that Turing machine is capable computing any computing function. Turing also showed that there were some functions that no turing machine can compute.

Theory of NP Complete pioneered by Steeve Cook and Richard Karp. Work in AI has helped explain why some instances of NP Complete problems are hard.

The great contribution of mathematics in AI is theory of Probability. Bayes rule underlies most modern approaches in AI.

Decision theory which combines probability theory with utility theory provides a formal and complete framework for decisions The work of Richard Bellman (1957) formalized a class of sequential decision problems called Markov decision processes.

Camillo Golgi (1843-1926) developed a staining technique allowing the observation of individual neurons in the brain. Nicholas Rashevsky (1938) was the first to apply mathematical models to the study of nervous system.

For artificial Intelligence to succeed we need two things: intelligence and an artifact. Charles babbage developed analytical engine consists of addressable memory, stored programs and conditional jumps and was the first artifact capable of universal computation. Ada Lovelace was the worlds first programmer.

AI has pioneered many ideas that have made their way back to mainstream computer science, including time sharing, interactive interpreters, personal computers with rapid development environments

Gestation and Birth of AI:

The first work of AI was done by Waren Mc Culloch and Walter Pitts, Knowledge of basic physiology and function of neurons in the brain, formal analysis of propositional logic due to Russell and Whitehead. Mc culloch and Pitts also suggested that suitably defined networks could learn. Donald Hebb (1949) demonstrated a simple updating rule for modifying the connection strengths between neurons. Minsky studied universal computation in neural networks. Alan Turing introduced the turing Test, machine learning, genetic algorithms, Reinforcement learning.

John Mc Carthy was first coined the term Artificial Intelligence. Simon Claimed " We have invented a computer program capable of thinking non numerically thereby solved the venerable mind body problem"

AI is the only field to attempt to build machinesthat function autonomously in complex and changing environments.

Newell and Simon's early success was followed up with the General Problem Solver or GPS.Newell and Simon to formulate the famous physical symbol system hypothesis which states that " A physical symbol system has the necessary and sufficient means general intelligent action."

The perceptron convergence theorem says that the learning algorithm can adjust the connection strengths of the perceptron to match any input data, provided such a match exists.

Minsky and papert book perceptrons proved that it could be shown to learn anything they were capable of representing they could represent little. Bryson and Ho, 1969 proposed new back propagation algorithm

Judea Pearl's (1988) probabilistic reasoning in Intelligent Systems led to a new acceptance of probability and decision theory in AI. Kurzweil (2005) writes, many thousands of AI applications are deeply embedded in the infrastructure of every industry.

Scope & current status:

Robotic Vehicles: A driverless robotic car named STANLEY sped through the rough terrain of the Mojave dessert at 22mph, finishing the 132-mile course first to win the 2005 DARPA grand challenge. Stanley is a Volkswagen Touareg outfitted with cameras, radar and laser rangefinders to sense the environment and on board software to command the steering, braking and acceleration

Speech recognition: A traveler calling united airlines to book a flight can have the entire conversation guided by an automated speech recognition and dialog management system.

Autonomous planning and Scheduling: Remote Agent program became the first on board autonomous planning program to control the scheduling of operations for a spacecraft. Remote Agent generated plans from high level goals specified from the ground and monitored the execution of those plans detecting, diagnosing and recovering from problems as they occurred.

Game Playing: IBM Deep Blue became the first computer program to defeat the world champion in a chess match when it bested Garry Kasparov. Kasparov said that new kind of intelligence

Spam fighting: Learning algorithms classify over a billion messages as spam, saving the recipient from having to waste time deleting what, for many users, could comprise 80% or 90% of all messages, if not classified away by algorithms.

Logistic Planning: US forces deployed a dynamic analysis and replanning Tool, DART to do automated logistic planning and scheduling for transportation. The AI planning techniques generating in hours a plan that would have taken weeks with older methods.

Robotics: The iRobot Corporation has sold over two million Roomba robotic vacuum cleaners for home use.

Machine Translation: A Computer program automatically translates from Arabic to English, allowing an English speaker to see headline “Ardogan Confirms that turkey would not accept any pressure, urging them to recognize cyprus”. The program uses a statistical model built from examples of Arabic to English Translations and from examples of English text totalling two trillion words.

Agents by describing *behavior*—the action that is performed after any given sequence of percepts. The job of AI is to design an agent program that implements the agent function— the mapping from percepts to actions.
computing device with physical sensors and actuators—we call this the architecture:

agent = architecture + program, current percept as input from the sensors and return an action to the actuators.
the agent program, which takes the current percept as input, and the agent function, which takes the entire percept history.
The agent program takes just the current percept as input because nothing more is available from the environment; if the agent's actions need to depend on the entire percept sequence, the agent will have to remember the percepts

Environments:
We must think about task environments, which are essentially the "problems" to which rational agents are the "solutions." How to specify a task environment, illustrating the process with a number of examples.

In our discussion of the rationality of the simple vacuum-cleaner agent, we had to specify the performance measure, the environment, and the agent's actuators and sensors
we call PEAS (Performance, Environment, Actuators, Sensors) description. In designing an agent, the first step must always be to specify the task environment as fully as possible.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Figure 2.4 PEAS description of the task environment for an automated taxi.

The actuators for an automated taxi include those available to a human driver: control over the engine through the accelerator and control over steering and braking. In addition, it will need output to a display screen or voice synthesizer to talk back to the passengers, and perhaps some way to communicate with other vehicles, politely or otherwise.

The basic sensors for the taxi will include one or more controllable video cameras so that it can see the road; it might augment these with infrared or sonar sensors to detect distances to other cars and obstacles. To avoid speeding tickets, the taxi should have a speedometer, and to control the vehicle properly, especially on curves, it should have an accelerometer. To determine the mechanical state of the vehicle, it will need the usual array of engine, fuel, and electrical system sensors. Like many human drivers, it might want a global positioning system (GPS) so that it doesn't get lost. Finally, it will need a keyboard or microphone for the passenger to request a destination.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers

Properties of task environments:

Fully observable vs. partially observable: If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable. A task environment is effectively fully observable if the sensors detect all aspects that are *relevant* to the choice of action; relevance, in turn, depends on the performance measure. Fully observable environments are convenient because the agent need not maintain any internal state to keep track of the world.
An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data—for example, a vacuum agent with only a local dirt sensor cannot tell whether there is dirt in other squares, and an automated taxi cannot see what other drivers are thinking. If the agent has no sensors at all then the environment is unobservable.

Single agent vs. multiagent: The distinction between single-agent and multiagent environments may seem simple enough. For example, an agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a two agent environment. There are, however, some subtle issues. First, we have described how an entity *may* be viewed as an agent, but we have not explained which entities *must* be viewed as agents.
Does an agent A (the taxi driver for example) have to treat an object B (another vehicle) as an agent, or can it be treated merely as an object behaving according to the laws of physics, analogous to waves at the beach or leaves blowing in the wind? The key distinction is whether B's behavior is best described as maximizing a performance measure whose value depends on agent A's behavior. For example, in chess, the opponent entity B is trying to maximize its performance measure, which, by the rules of chess, minimizes agent A's performance measure. Thus, chess is a competitive multiagent environment. In the taxi-driving environment, on the other hand, avoiding collisions maximizes the performance measure of all agents, so it is a partially cooperative multiagent environment. It is also partially competitive because, for example, only one car can occupy a parking space. The agent-design problems in multiagent environments are often quite different from those in single-agent environments; for example, communication often emerges as a rational behavior in multiagent environments; in some competitive environments, randomized behavior is rational because it avoids the pitfalls of predictability.

Deterministic vs. stochastic. If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic; otherwise, it is stochastic. In principle, an agent need not worry about uncertainty in a fully observable, deterministic environment. If the environment is partially observable, however, then it could *appear* to be stochastic. Most real situations are so complex that it is impossible to keep track of all the unobserved aspects; for practical purposes, they must be treated as stochastic.

Our use of the word "stochastic" generally implies that uncertainty about outcomes is quantified in terms of probabilities; a nondeterministic environment is one in which actions are characterized by their *possible* outcomes, but no probabilities are attached to them. Nondeterministic environment descriptions are usually associated with performance measures that require the agent to succeed for *all possible* outcomes of its actions.

Episodic vs. sequential: In an episodic task environment, the agent's experience is divided into atomic episodes. In each episode the agent receives a percept and then performs a single action. Crucially, the next episode does not depend on the actions taken in previous episodes. Many classification tasks are episodic. For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions;
In sequential environments, on the other hand, the current decision could affect all future decisions. Chess and taxi driving are sequential: in both cases, short-term actions can have long-term consequences. Episodic environments are much simpler than sequential environments because the agent does not need to think ahead.

Static vs. dynamic: If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise, it is static. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time. Dynamic environments, on the other hand, are continuously asking the agent what it wants to do; if it hasn't decided yet, that counts as deciding to do nothing. If the environment itself does not change with the passage of time but the agent's performance score does, then we say the environment is semidynamic.

Discrete vs. continuous: The discrete/continuous distinction applies to the *state* of the environment, to the way *time* is handled, and to the *percepts* and *actions* of the agent. For example, the chess environment has a finite number of distinct states (excluding the clock). Chess also has a discrete set of percepts and actions. Taxi driving is a continuous-state and continuous-time problem: the speed and location of the taxi and of the other vehicles sweep through a range of continuous values and do so smoothly over time. Taxi-driving actions are also continuous (steering angles, etc.). Input from digital cameras is discrete, strictly speaking, but is typically treated as representing continuously varying intensities and locations.

Known vs. unknown: Strictly speaking, this distinction refers not to the environment itself but to the agent's (or designer's) state of knowledge about the "laws of physics" of the environment. In a known environment, the outcomes (or outcome probabilities if the environment is stochastic) for all actions are given. Obviously, if the environment is unknown, the agent will have to learn how it works in order to make good decisions. Note that the distinction between known and unknown environments is not the same as the one between fully and partially observable environments. It is quite possible for a *known* environment to be *partially* observable—for example, in solitaire card games, I know the rules but am still unable to see the cards that have not yet been turned over. Conversely, an *unknown* environment can be *fully* observable—in a new video game, the screen may show the entire game state but I still don't know what the buttons do until I try them.

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
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Each kind of agent program combines particular components in particular ways to generate actions.

Simple reflex agents

The simplest kind of agent is the **simple reflex agent**. These agents select actions on the basis of the *current* percept, ignoring the rest of the percept history. For example, the vacuum agent whose agent function is tabulated is a simple reflex agent, because its decision is based only on the current location and on whether that location contains dirt. the vacuum agent program is very small indeed compared to the corresponding table.

Simple reflex behaviors occur even in more complex environments. Imagine yourself as the driver of the automated taxi. If the car in front brakes and its brake lights come on, then you should notice this and initiate braking. In other words, some processing is done on the visual input to establish the condition we call "The car in front is braking." Then, this triggers some established connection in the agent program to the action "initiate braking." We call such a connection a **condition–action rule** written as

if *car-in-front-is-braking* **then** *initiate-braking*.

Humans also have many such connections, some of which are learned responses (as for driving) and some of which are innate reflexes (such as blinking when something approaches the eye).

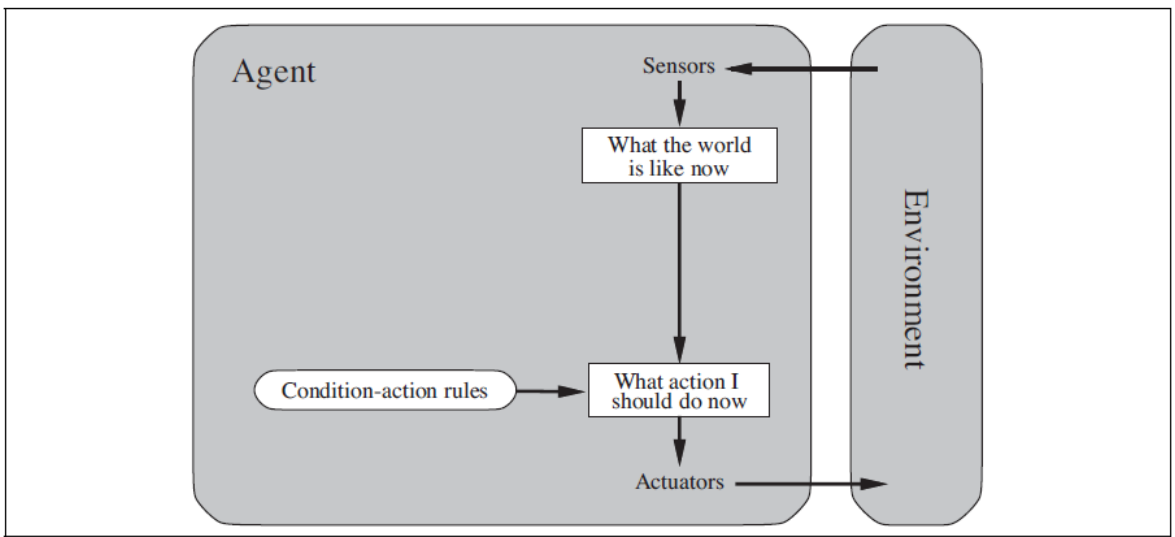


Figure 2.9 Schematic diagram of a simple reflex agent.

function SIMPLE-REFLEX-AGENT(percept) **returns** an action
persistent: rules, a set of condition–action rules
state←INTERPRET-INPUT(percept)
rule←RULE-MATCH(state, rules)
action ←rule.ACTION
return action

Fig: A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept

A simple reflex agent driving behind such a car would either brake continuously and unnecessarily, or, worse, never brake at all.

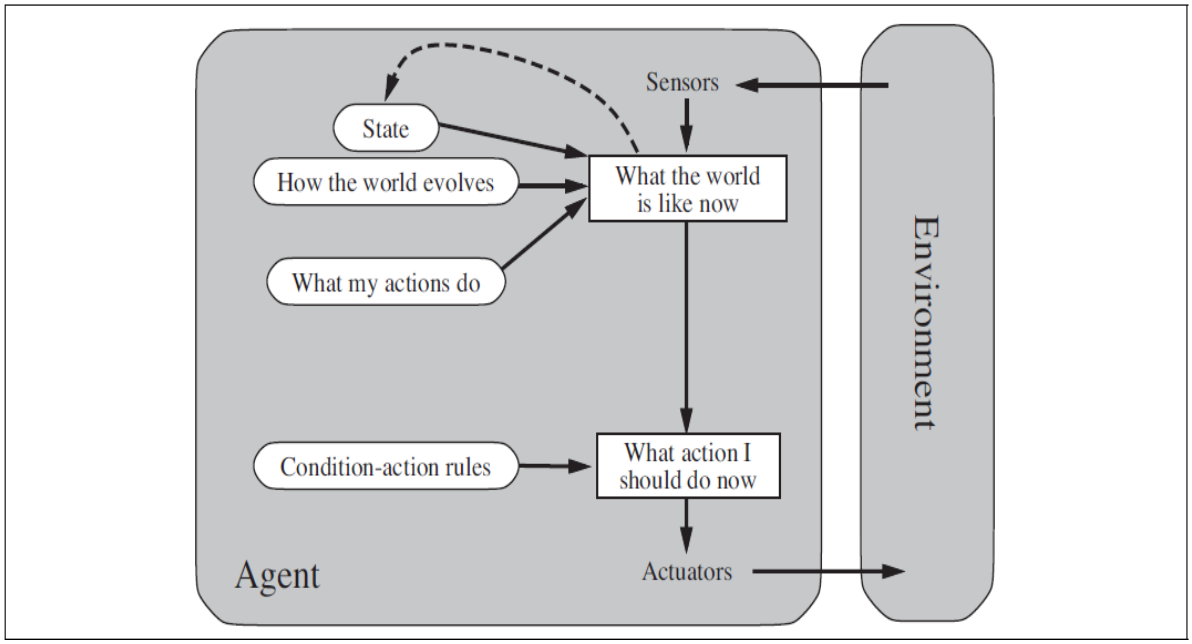
We can see a similar problem arising in the vacuum world. Suppose that a simple reflex vacuum agent is deprived of its location sensor and has only a dirt sensor. Such an agent has just two possible percepts: [Dirty] and [Clean]. It can Suck in response to [Dirty]; what should it do in response to [Clean]? Moving Left fails (forever) if it happens to start in square A, and moving Right fails (forever) if it happens to start in square B. Infinite loops are often unavoidable for simple reflex agents operating in partially observable environments.

For example,
if the vacuum agent perceives [Clean], it might flip a coin to choose between Left and Right . It is easy to show that the agent will reach the other square in an average of two steps. Then, if that square is dirty, the agent will clean it and the task will be complete. Hence, a randomized simple reflex agent might outperform a deterministic simple reflex agent.

2. **Model-based reflex agents**

The most effective way to handle partial observability is for the agent to *keep track of the part of the world it can't see now*. That is, the agent should maintain some sort of **internal state** that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state. For the braking problem, the internal state is not too extensive—just the previous frame from the camera, allowing the agent to detect when two red lights at the edge of the vehicle go on or off simultaneously. For other driving tasks such as changing lanes, the agent needs to keep track of where the other cars are if it can't see them all at once. And for any driving to be possible at all, the agent needs to keep track of where its keys are.
Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program. First, we need some information about how the world evolves independently of the agent—for example, that an overtaking car generally will be closer behind than it was a moment ago. Second, we need some information about how the agent's own actions affect the world—for example, that when the agent turns the steering wheel clockwise, the car turns to the right, or that after driving for five minutes northbound

on the freeway, one is usually about five miles north of where one was five minutes ago. This knowledge about "how the world works"—whether implemented in simple Boolean circuits or in complete scientific theories—is called a **model** of the world. An agent that uses such a



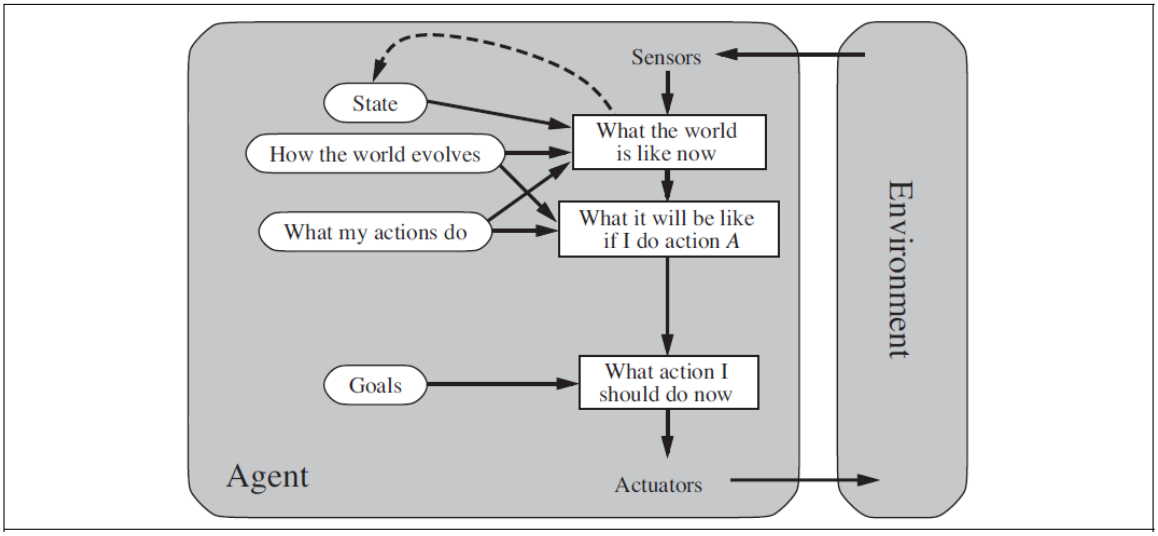
function MODEL-BASED-REFLEX-AGENT(percept) **returns** an action
persistent: state, the agent's current conception of the world state
model , a description of how the next state depends on current state and action
rules, a set of condition–action rules
action, the most recent action, initially none
state←UPDATE-STATE(state, action, percept ,model)
rule←RULE-MATCH(state, rules)
action ←rule.ACTION
return action

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.

Goal-based agents:
Knowing something about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to. In other words, as well as a current state description, the GOAL agent needs some sort of **goal** information that describes situations that are desirable—for example, being at the passenger's destination. The agent program can combine this with the model (the same information as was used in the modelbased reflex agent) to choose actions that achieve the goal.

Sometimes goal-based action selection is straightforward—for example, when goal satisfaction results immediately from a single action. Sometimes it will be more tricky—for example, when the agent has to consider long sequences of twists and turns in order to find a

way to achieve the goal.



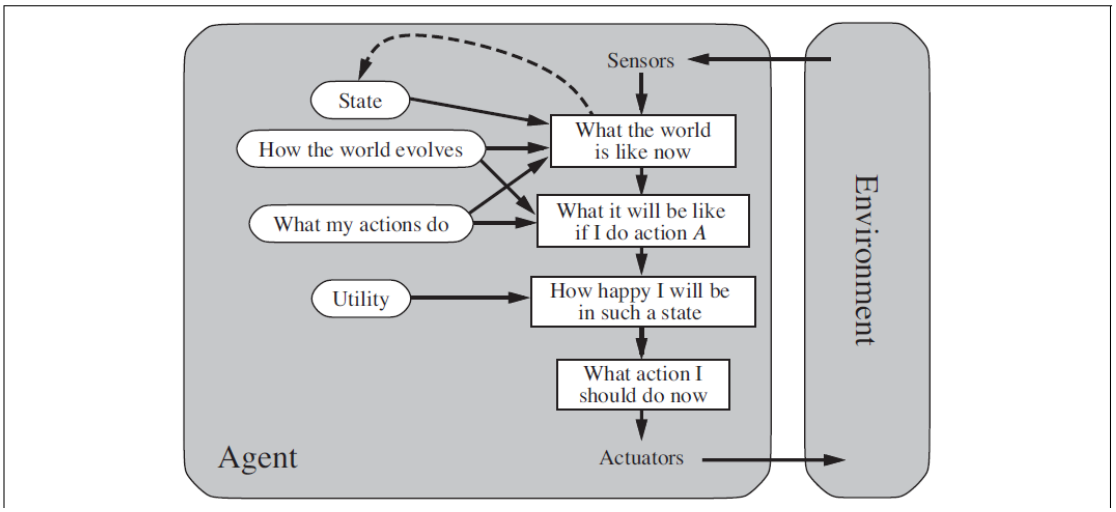
A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.
A goal-based agent, in principle, could reason that if the car in front has its brake lights on, it will slow down. Given the way the world usually evolves, the only action that will achieve the goal of not hitting other cars is to brake.

the goal-based agent appears less efficient, it is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified. If it starts to rain, the agent can update its knowledge of how effectively its brakes will operate; this will automatically cause all of the relevant behaviors to be altered to suit the new conditions. The goal-based agent's behavior can easily be changed to go to a different destination, simply by specifying that destination as the goal.

Utility-based agents

Goals alone are not enough to generate high-quality behavior in most environments. For example, many action sequences will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others. Goals just provide a crude binary distinction between "happy" and "unhappy" states. A more general performance measure should allow a comparison of different world states according to exactly how happy they would make the agent. Because "happy" does not sound very scientific, economists and computer scientists use the term **utility** instead.

An agent's **utility function** is essentially an internalization
of the performance measure. If the internal utility function and the external performance measure are in agreement, then an agent that chooses actions to maximize its utility will be rational according to the external performance measure. First, when there are conflicting goals,
only some of which can be achieved (for example, speed and safety), the utility function specifies the appropriate tradeoff. Second, when there are several goals that the agent can aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighed against the importance of the goals.
Partial observability and stochasticity are ubiquitous in the real world, and so, therefore, is decision making under uncertainty. Technically speaking, a rational utility-based agent EXPECTED UTILITY chooses the action that maximizes the **expected utility** of the action outcomes—that is, the
utility the agent expects to derive, on average, given the probabilities and utilities of each outcome.



A model-based, utility-based agent. It uses a model of the world, along with
a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

A learning agent can be divided into four conceptual components. The most important distinction is between LEARNING ELEMENT the **learning element**, which is responsible for making improvements, and the **performance element**, which is responsible for selecting external actions. The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions. The learning element uses feedback from the **critic** on how the agent is doing and determines how the performance element should be modified to do better in the future. The critic tells the learning element how well the agent is doing with respect to a fixed
performance standard. The critic is necessary because the percepts themselves provide no indication of the agent's success

The last component of the learning agent is the **problem generator**. It is responsible for suggesting actions that will lead to new and informative experiences. The point is that if the performance element had its way, it would keep doing the actions that are best, given what it knows. But if the agent is willing to explore a little and do some perhaps suboptimal actions in the short run, it might discover much better actions for the long run.

State Space Representation

Goal-based agents that use more advanced **factored** or **structured** representations are usually called **planning agents**

uninformed search algorithms—algorithms that are given no information about the problem other than its definition. Although some of these algorithms can solve any solvable problem, none of them can do so efficiently. **Informed** search algorithms, on the other hand, can do quite well given some guidance on where to look for solutions.

Goals help organize behavior by limiting the objectives that the agent is trying to achieve and hence the actions it needs to consider. **Goal formulation**, based on the current situation and the agent's performance measure, is the first step in problem solving.

Problem formulation is the process of deciding what actions and states to consider, given a goal.

The process of looking for a sequence of actions that reaches the goal is called **search**.

A search algorithm takes a problem as input and returns a **solution** in the form of an action sequence.

Once a solution is found, the actions it recommends can be carried out. This is called the **execution** phase.

After formulating a goal and a problem to solve, the agent calls a search procedure to solve it. It then uses the solution to guide its actions, doing whatever the solution recommends as the next thing to do—typically, the first action of the sequence—and then removing that step from the sequence. Once the solution has been executed, the agent will formulate a new goal.

An agent that carries out its plans with its eyes closed, so to speak, must be quite certain of what is going on. Control theorists call this an **open-loop** system, because ignoring the percepts breaks the loop between agent and environment.

Formulating problems

A formulation of the problem of getting of the initial state, actions, transition model, goal test, and path cost.

Well-defined problems and solutions

A **problem** can be defined formally by five components:

The **initial state** that the agent starts in.

A description of the possible **actions** ACTIONS available to the agent. Given a particular state s, ACTIONS(s) returns the set of actions that can be executed in s. We say that each of these actions is **applicable** in s.

A description of what each action does; the formal name for this is the **transition model**, specified by a function RESULT(s, a) that returns the state that results from doing action a in state s. We also use the term **successor** to refer to any state reachable from a given state by a single action Together, the initial state, actions, and transition model implicitly define the **state space** of the problem—the set of all states reachable from the initial state by any sequence of actions. The state space forms a directed network or **graph** in which the nodes are states and the links between nodes are actions. A **path** in the state space is a sequence of states connected by a sequence of actions.

The **goal test**, which determines whether a given state is a goal state. Sometimes there

is an explicit set of possible goal states, and the test simply checks whether the given state is one of them.
A **path cost** function that assigns a numeric cost to each path. The problem-solving agent chooses a cost function that reflects its own performance measure.
A **solution** to a problem is an action sequence that leads from the initial state to a goal state. Solution quality is measured by the cost function, and an **optimal solution** has the lowest path cost among all solutions.
path cost function, and an **optimal solution** has the lowest path cost among all solutions.

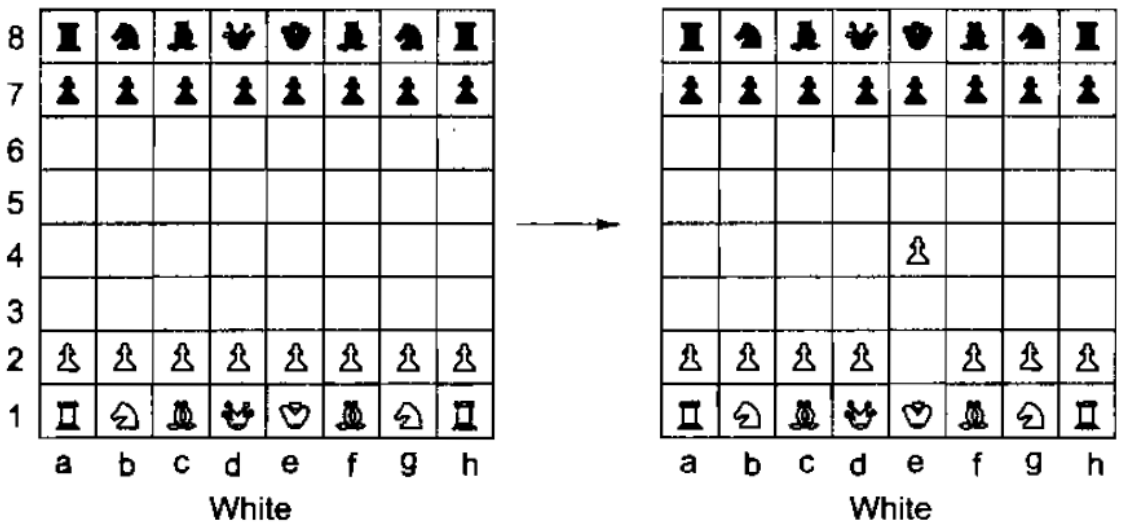
To build a system a particular problem, we need to do four things:

1. Define the problem precisely. The initial situation will be as as well as what final situation constitute acceptable solution to the problem
2. Analyze the problem : finding the possible techniques to solve the problem
3. Isolate and represent the task knowledge that is necessary to solve the problem
4. Choose the best solving technique apply it to particular problem

Chess game:
Build a chess program that would specifies the starting position and the legal moves and the winning position of the game.
The starting position can be described as 8X8 array each position contains a symbol standing for the appropriate piece. The winning position or goal state in which the opponent does not have a legal move and his or her king is under attack q

Formal description of a problem:

1. Define a state space that contain all possible configurations of the relevant objects
2. Specify one or more states within that space that describe possible situations from which the problem solving process may start. These states are called initial states
3. Specify one or more states that would be acceptable as solutions to the problems These are called goal states
4. Specify the set of rules that describe the actions
   i) Assumptions are present in the informal description
   ii) How general should the rules be
   iii) Work required to solve the problem should be precomputed and represent the rules



Legal move:

White pawn at Square(file e, rank 2) AND Square(file e, rank 3) is empty AND Square(file e, rank 4) is empty	→	move pawn from Square(file e, rank 2) to Square(file e, rank 4)

States: Any arrangement of 0 to 8 queens on the board is a state.
• **Initial state**: No queens on the board.
• **Actions**: Add a queen to any empty square.
• **Transition model**: Returns the board with a queen added to the specified square.
• **Goal test**: 8 queens are on the board, none attacked.

States: All possible arrangements of n queens ($0 \leq n \leq 8$), one per column in the leftmost n columns, with no queen attacking another.
• **Actions**: Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.

Water Jug problem:
Two jugs are given, a 4 gallon one and a 3 gallon one Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4 gallon jug?

Gallons in the 4-Gallon Jug	Gallons in the 3-Gallon Jug	Rule Applied
0	0	
		2
0	3	
		9
3	0	
		2
3	3	
		7
4	2	
		5 or 12
0	2	
		9 or 11
2	0	

Legal Moves:

Solution

Toy problems

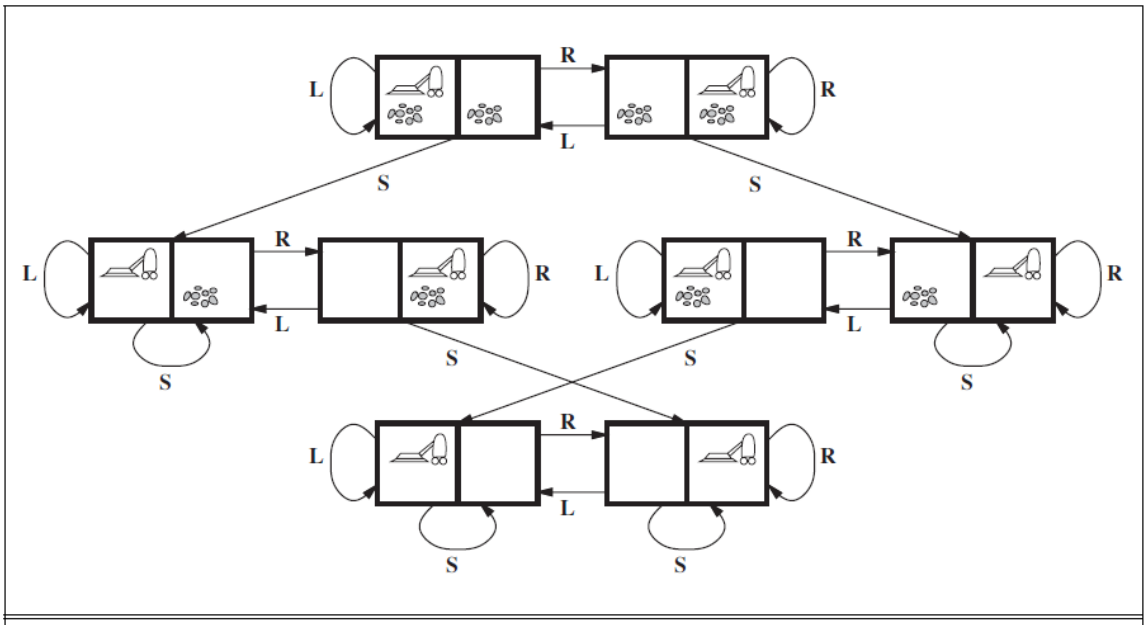


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

States: The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt. Thus, there are $2 \times 2^2 = 8$ possible world states. A larger environment with n locations has $n \cdot 2^n$ states.

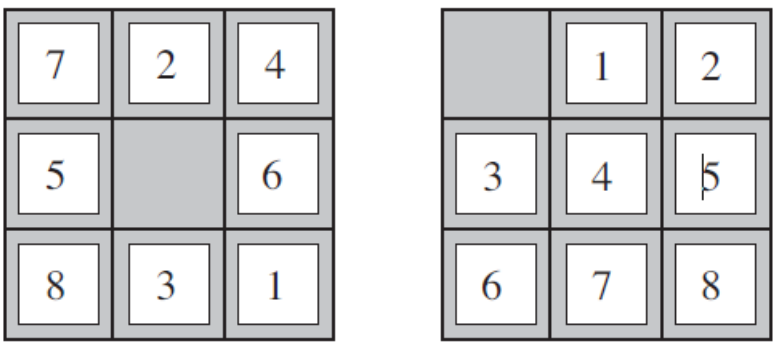
• **Initial state**: Any state can be designated as the initial state.

• **Actions**: In this simple environment, each state has just three actions: *Left*, *Right*, and *Suck*. Larger environments might also include *Up* and *Down*.

• **Transition model**: The actions have their expected effects, except that moving *Left* in the leftmost square, moving *Right* in the rightmost square, and *Suck*ing in a clean square have no effect.

• **Goal test**: This checks whether all the squares are clean.

8-Puzzle:
consists of a 3×3 board with eight numbered tiles and a blank space.



States: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
• **Initial state**: Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states.
• **Actions**: The simplest formulation defines the actions as movements of the blank space *Left*, *Right*, *Up*, or *Down*. Different subsets of these are possible depending on where the blank is.
• **Transition model**: Given a state and action, this returns the resulting state if we apply *Left* to the start state in Figure 3.4, the resulting state has the 5 and the blank switched.
• **Goal test**: This checks whether the state matches the goal configuration
• **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

Tree Search and Graph Search:

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

```
function GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
            only if not in the frontier or explored set
```

The way to avoid exploring redundant paths is to remember where one has been. To do this, we augment the TREE-SEARCH algorithm with a data structure called the **explored.**

GRAPH-SEARCH algorithm contains at most one copy of each state, so we can think of it as growing a tree directly on the state-space graph. The algorithm has another nice property: the frontier **separates** the state-space graph into the explored region and the unexplored region, so that every path from the initial state to an unexplored state has to pass through a state in the frontier.