Artificial Intelligence [ETCS-310]

Dr. A K Yadav
Amity School of Engineering and Technology
(affiliated to GGSIPU, Delhi)
akyadav1@amity.edu
akyadav@akyadav.in
www.akyadav.in
+91 9911375598

April 30, 2020

Introduction to Artificial Intelligence

- Acting humanly: The Turing Test approach
  "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 are better." (Rich and Knight, 1991)
  To pass the Turing Test, the computer would need to possess the following capabilities:
  1. natural language processing to enable it to communicate successfully in English
  2. knowledge representation to store what it knows or hears
  3. automated reasoning to use the stored information to answer questions and to draw new conclusions
  4. machine learning to adapt to new circumstances and to detect and extrapolate patterns
  5. computer vision to perceive objects

6 robotics to manipulate objects and move about.

- Thinking humanly: The cognitive modeling approach
  "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 with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)
  3 way to find out how human thinks:
  1. through introspection—trying to catch our own thoughts as they go by
  2. through psychological experiments—observing a person in action
  3. through brain imaging—observing the brain in action
  - Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program.

- If the program's input–output behavior matches corresponding human behavior, that is evidence that some of the program's mechanisms could also be operating in humans.

The interdisciplinary field of cognitive science brings together computer models from AI and experimental techniques from psychology to construct precise and testable theories of the human mind.

- Thinking rationally: The "laws of thought" approach
"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985)
"The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
  - The laws of thought were supposed to govern the operation of the mind; their study initiated the field called logic.
  - Logicians in the 19th century developed a precise notation for statements about all kinds of objects in the world and the relations among them

- By 1965, programs existed that could, in principle, solve any solvable problem described in logical notation.

There are two main obstacles to this approach:

1. It is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain
2. There is a big difference between solving a problem "in principle" and solving it in practice. Even problems with just a few hundred facts can exhaust the computational resources of any computer unless it has some guidance as to which reasoning steps to try first.

- Acting rationally: The rational agent approach

  "Computational Intelligence is the study of the design of intelligent agents." (Poole et al., 1998)
  "AI . . . is concerned with intelligent behaviour in artifacts." (Nilsson, 1998)
  - An agent is just something that acts.
  - all computer programs do something,

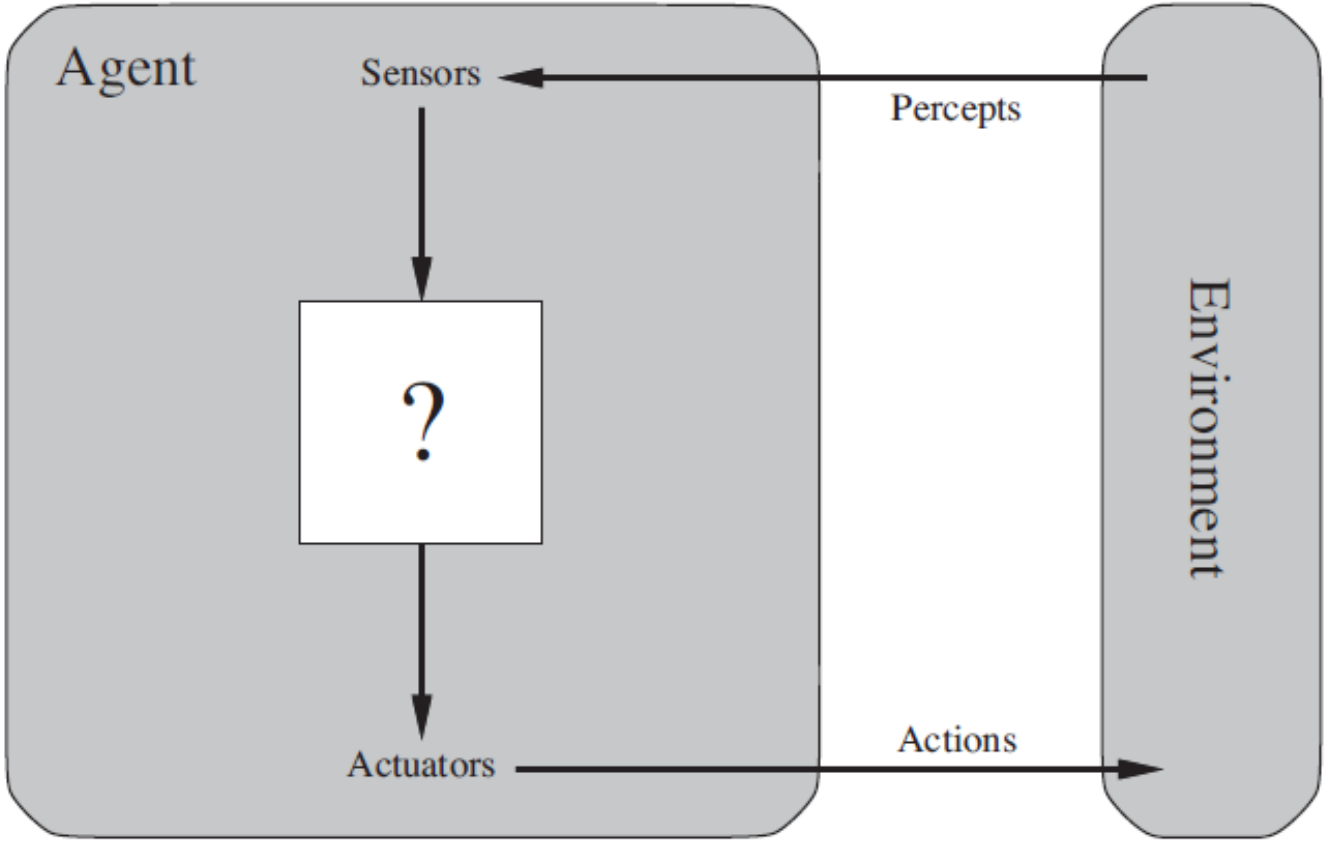
- but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals.
- A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

Applications of Artificial Intelligence

- Industry
- Finance
- Medicine
- Science
- Games
- Driver lass car
- Speech Recognition

Intelligent Agents

- Anything that can gather information about its environment and take action based on that information.
- Systems that can reasonably be called intelligent.
- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.



Examples:

- A **human agent** has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators.
- A **robotic agent** might have cameras and infrared range finders for sensors and various motors for actuators.

- A **software agent** receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.
- **percept** refer to the agent's perceptual inputs at any given instant.
- An agent's percept sequence is the complete history of everything the agent has ever perceived.
- an agent's choice of action at any given instant can depend on the entire percept sequence observed to date, but not on anything it hasn't perceived.
- Mathematically speaking, we say that an agent's behavior is described by the agent function that maps any given percept sequence to an action.
- Internally, the agent function for an artificial agent will be implemented by an agent program.

- The difference between two is : The agent function is an abstract mathematical description; the agent program is a concrete implementation, running within some physical system.
- Vacuum-cleaner or Fish Game etc.
- agent function: if the current square is dirty, then suck; otherwise, move to the other square.

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

Good Behavior

- A rational agent is one that does the right thing.
- When an agent is setuped in an environment, it generates a sequence of actions according to the percepts it receives
- This sequence of actions causes the environment to go through a sequence of states
- If the sequence is desirable, then the agent has performed well
- This desirability is used to measure performance that evaluates any given sequence of environment states.
- We use environment states, not agent states in performance measure.
- there is not one fixed performance measure for all tasks and agents; typically, a designer will devise one appropriate environment as per the circumstances
- Examples: Vacuum Cleaner, NIFR ranking, army, etc.

Rationality

What is rational at any given time depends on four things:

- The performance measure that defines the criterion of success.
- The agent's prior knowledge of the environment.
- The actions that the agent can perform.
- The agent's percept sequence to date.

Definitaion 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.
Is vacuum-cleaner a rational agent?
To answer this first we have to define

- what is the performance measure?
- what is known about the environment?
- what sensors and actuators the agent has?

Omniscience, learning, and autonomy

- We need to be careful to distinguish between rationality and omniscience.
- An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality.
- Examples
- Rationality is not the same as perfection. Rationality maximizes expected performance, while perfection maximizes actual performance.
- Our definition of rationality does not require omniscience, because the rational choice depends only on the percept sequence to date.

- Doing actions in order to modify future percepts—sometimes called information gathering— is an important part of rationality.
- information gathering is provided by the exploration
- A rational agent requires not only to gather information but also to learn as much as possible from what it perceives
- Dung beetle and sphex wasp
- Evolution has built an assumption into the beetle's behavior, and when it is violated, unsuccessful behavior results
- an agent relies on the prior knowledge of its designer rather than on its own percepts, we say that the agent lacks autonomy.
- A rational agent should be autonomous—it should learn what it can to compensate for partial or incorrect prior knowledge.
- It would be reasonable to provide an artificial intelligent agent with some initial knowledge as well as an ability to learn.

- After sufficient experience of its environment, the behavior of a rational agent can become effectively independent of its prior knowledge.
- Hence, the incorporation of learning allows one to design a single rational agent that will succeed in a vast variety of environments.

Specifying the task environment

- Performance
- Environment
- Actuators
- Sensors

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.
  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
  If the agent has no sensors at all then the environment is unobservable.
- **Single agent vs. multiagent:**

- **Deterministic vs. stochastic:**
  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.
  We say an environment is uncertain if it is not fully observable or not deterministic
- **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.

- **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.
  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
- **Known vs. unknown:**
  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.

As one might expect, the hardest case is partially observable, multiagent, stochastic, sequential, dynamic, continuous, and unknown.

Structure of the Agents

- Agent takes action according to sequence of percepts.
- What action is to be taken is decided through agent program.
- Percept is received by sensors and implementation of action is through actuators.
- So agent is a combination of some physical devices as sensors & actuators and software program as agent program.
  $agent = architecture + agent\ program$

Agent programs

- The all agent programs we design have the same skeleton
- They take the current percept as input from the sensors and return an action to the actuators
- The difference between the agent program and the agent function is first takes the current percept as input because nothing more is available from the environment and second takes the entire percept history.
- If the agent's actions need to depend on the entire percept sequence, the agent will have to remember the percepts.
- We describe the agent programs in the simple pseudo code language and not in a particular programming language.

- The example of a TABLE-DRIVEN-AGENT program is given below:

  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 $\leftarrow$ LOOKUP(*percepts*, *table*)

  return *action*

  The program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.
- Large size of the table is a challenge.
- But still it do what we want to do. It implements the desired agent function.

- The key challenge for AI is to find out how to write programs that, to the extent possible, produce rational behavior from a small program rather than from a vast table.

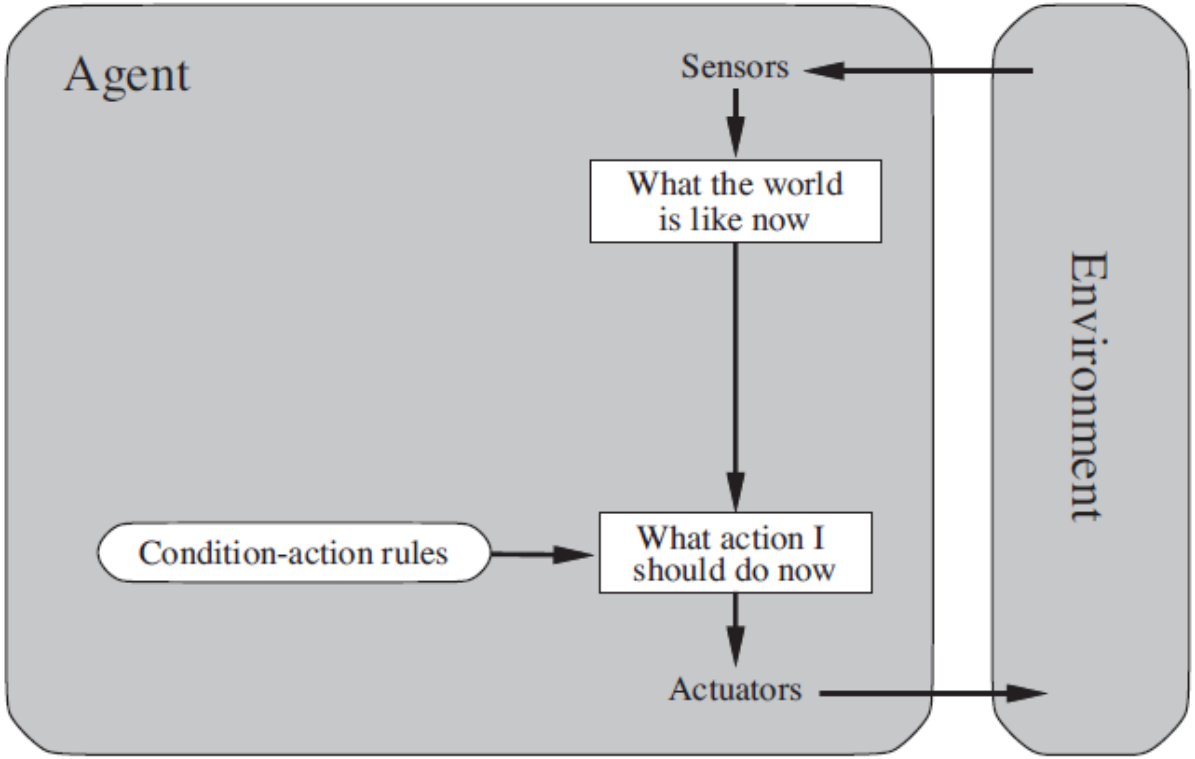
Type of Agents

There are four basic agent programs:

- Simple reflex agents
  The agent program for a simple reflex agent in the two-state vacuum environment is listed below
  - 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

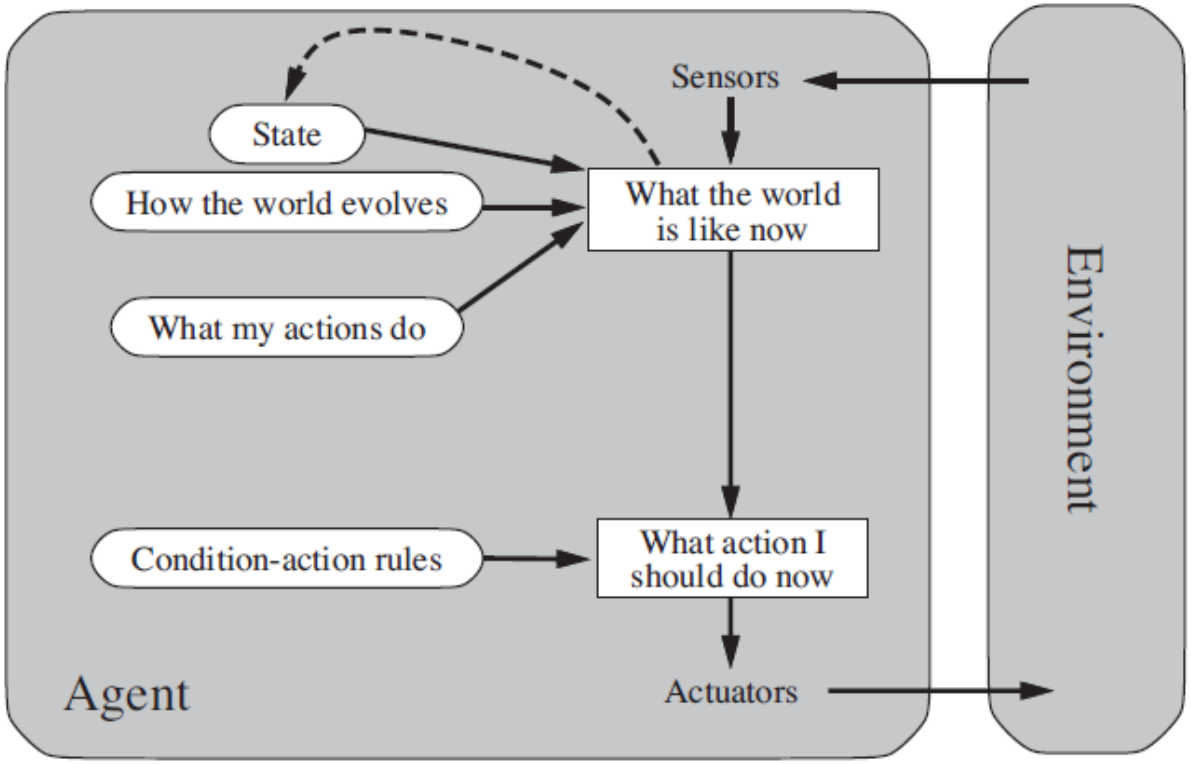
  A simple reflex agent acts according to a rule whose condition matches the current state, as defined by the percept as below
  - function SIMPLE-REFLEX-AGENT(percept) returns an action
  - persistent: rules, a set of condition–action rules
  - state←INTERPRET-INPUT(percept)
  - rule←RULE-MATCH(state, rules)
  - action ←rule.ACTION

- return action



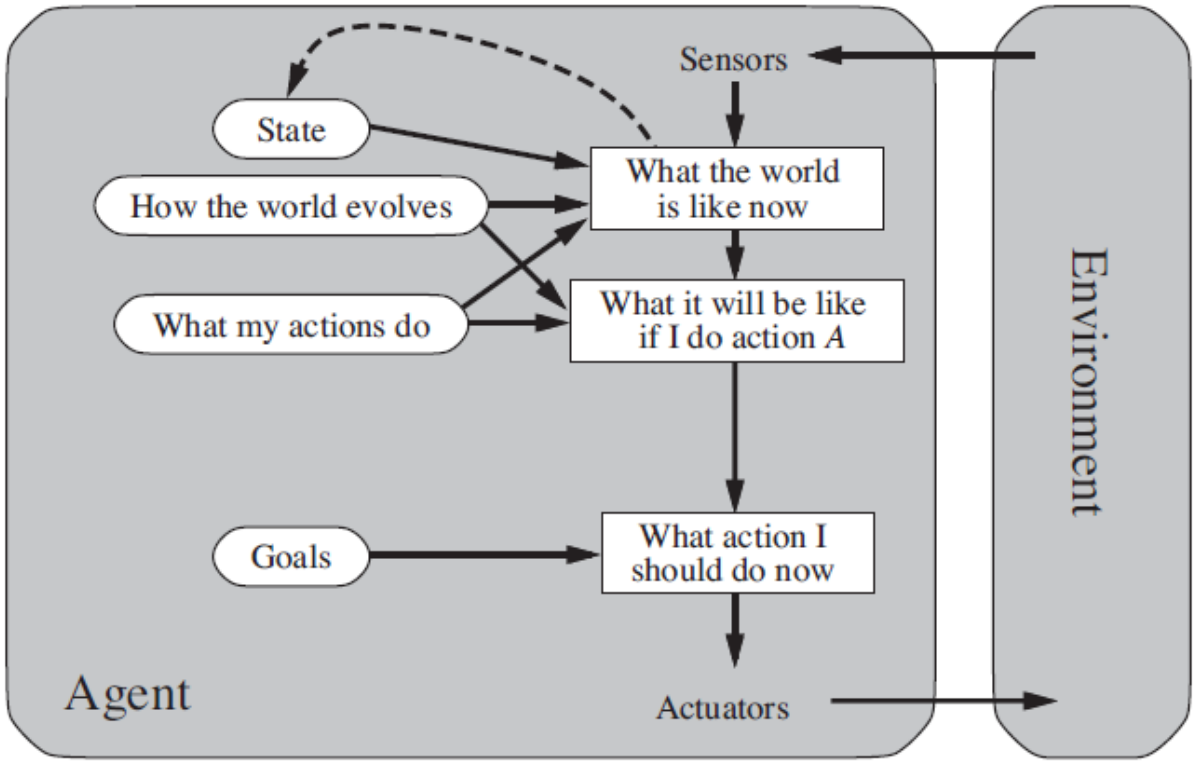
- Model-based reflex agents
  A model-based reflex agent keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.
  - function MODEL-BASED-REFLEX-AGENT(percept) returns an action

- 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


Sensors
State
How the world evolves
What the world is like now
What my actions do
Environment
Condition-action rules
What action I should do now
Agent
Actuators

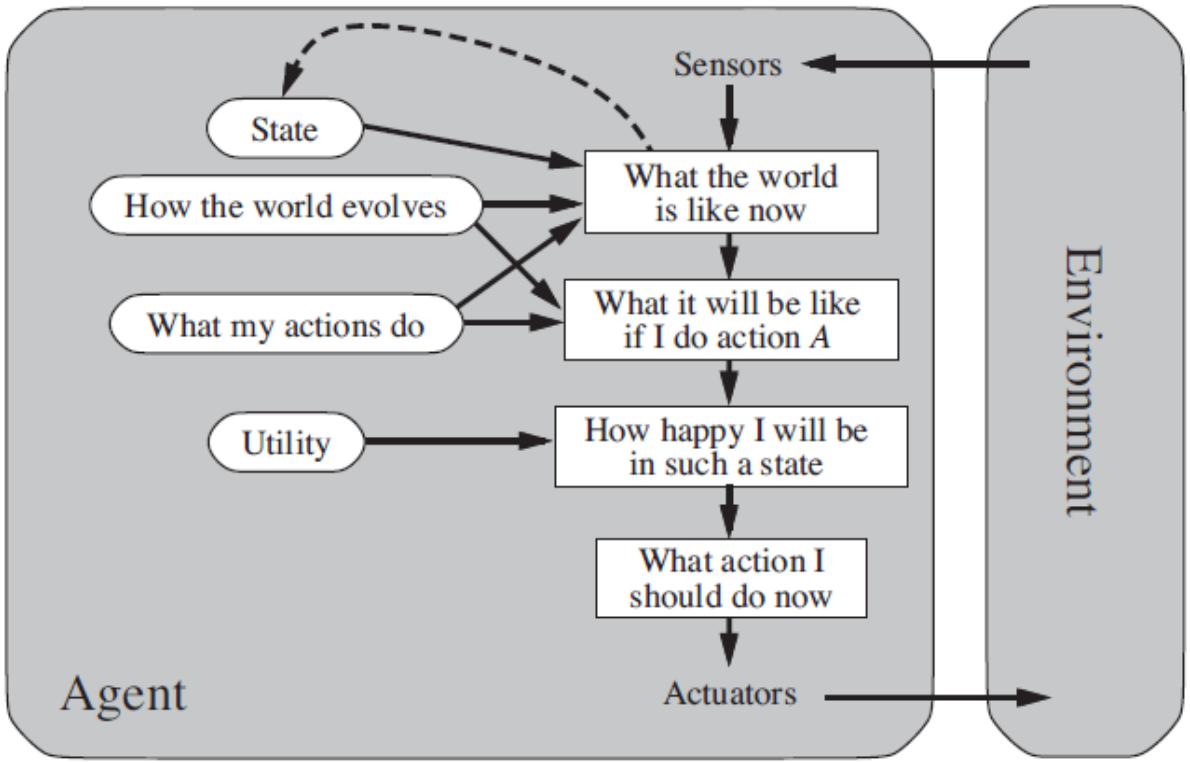
- Goal-based agents

  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.



- Utility-based agents

  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.



Problem Solving by searching: Problem-Solving Agents

- - **Goal formulation**
  - Goals help organize behavior by limiting the objectives that the agent is trying to achieve and hence the actions it needs to consider
  - Based on the current situation and the agent's performance measure, Goal formulation is the first step in problem solving
  - **Problem formulation**
  - The agent's task is to find out how to act, now and in the future, so that it reaches a goal state
  - It needs to decide what sorts of actions and states it should consider to achieve the goal.
  - Problem formulation is the process of deciding what actions and states to consider for a given a goal
- An agent with several immediate options of unknown value can decide what to do by first examining future actions that eventually lead to states of known value.

- Under some assumptions, the solution to any problem is a fixed sequence of actions
- 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
- A simple "formulate, search, execute" design for the agent is required.
- 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 and then removing that step from the sequence
- Once the solution has been executed, the agent will formulate a new goal

Formulating problems

A problem can be defined formally by five components:

- **initial state**:
  The initial state that the agent starts in.
- **actions**:
  A description of the possible actions available to the agent
- **transition model**:
  A description of what each action does;
- **goal test**:
  The goal test, which determines whether a given state is a goal state
- **path cost**:
  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 path cost function, and an optimal solution has the lowest path cost among all solutions.

Measuring problem-solving performance

- **Completeness:** Is the algorithm guaranteed to find a solution when there is one
- **Optimality:** Does the strategy find the optimal solution
- **Time complexity:** How long does it take to find a solution
- **Space complexity:** How much memory is needed to perform the search

Search Strategies

- **Uninformed search or blind search** Strategies have no additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from a non-goal state.
- **Informed search or heuristic search** Strategies that know whether one non-goal state is "more promising" than another are called informed search or heuristic search strategies.

Uninformed search or blind search

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening depth-first search
- Bidirectional search

Informed search or heuristic search

- In Informed search, additional knowledge of the problem is used find the next move in the search algorithm.
- can find solutions more efficiently as compared to uninformed strategy.
- The approach we consider is called best-first search in which a node is selected for expansion based on an evaluation function, f(n).
- The evaluation function is construed as a cost estimate, so the node with the lowest evaluation is expanded first.
- The implementation of best-first graph search is identical to that for uniform-cost search except for the use of f instead of g to order the priority queue.
- The choice of f determines the search strategy.

- Most best-first algorithms include a heuristic function h(n) as a component of f.
- h(n) = estimated cost of the cheapest path from the state at node n to a goal state.
- h(n) takes a node as input and it depends only on the state at that node
- Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm.
- h(n) is arbitrary, nonnegative, problem-specific functions, with one constraint: if n is a goal node, then h(n)=0.
-

Informed search algorithms

- Greedy best-first search
- A* search
- Memory-bounded heuristic search
  - Iterative Deepening A* (IDA*)
  - Recursive best-first search (RBFS)
  - Memory-bounded A* (MA*)
  - Simplified Memory-bounded A* (SMA*)

Greedy best-first search

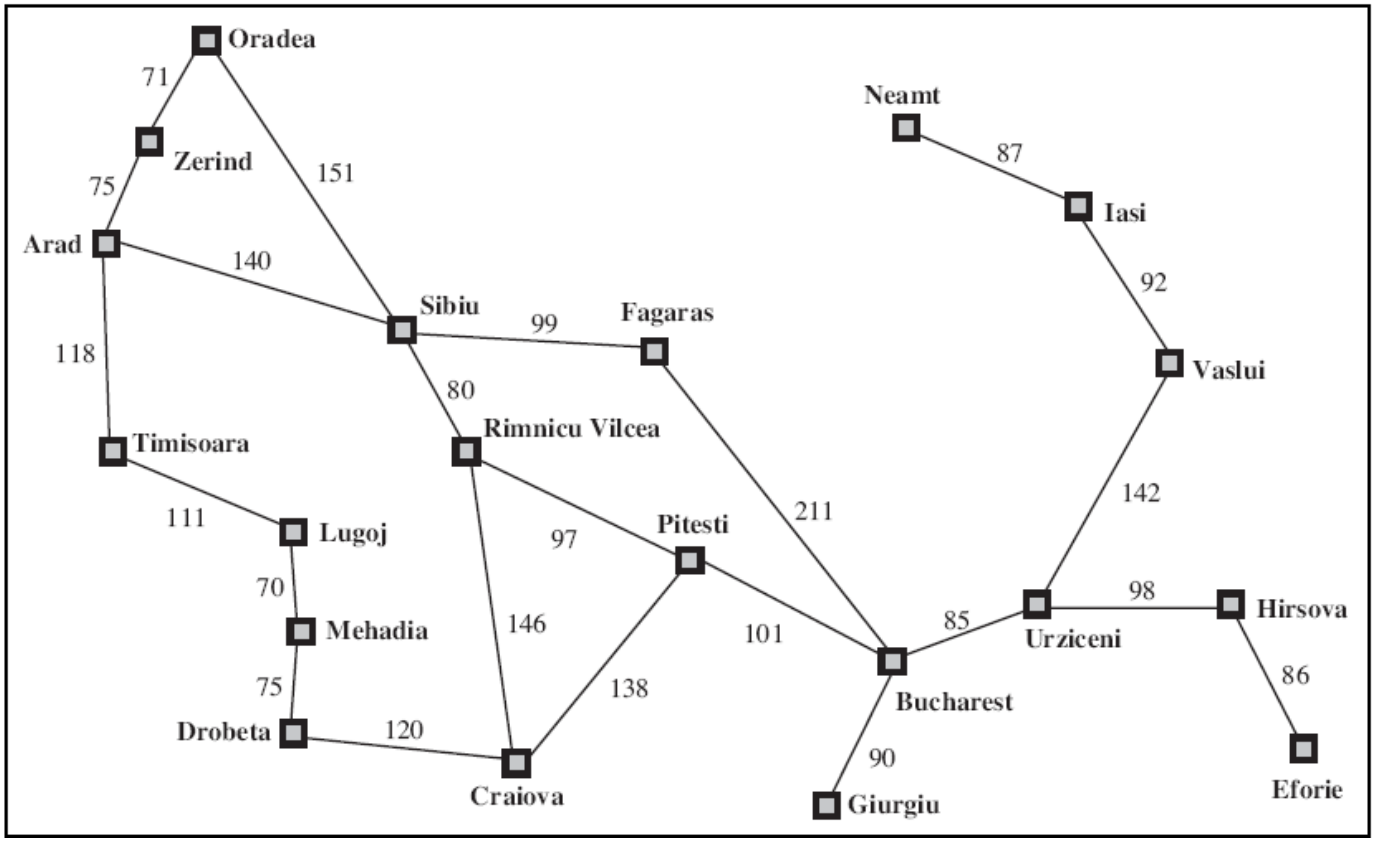
- expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.
- It evaluates nodes by using just the heuristic function; f(n) = h(n).
- Greedy best-first search using h finds a solution without ever expanding a node that is not on the solution path
- Its search cost is minimal
- It is not optimal
- Greedy best-first tree search is also incomplete even in a finite state space, much like depth-first search.
- The graph search version is complete in finite spaces, but not in infinite ones.
- The worst-case time and space complexity for the tree version is $O(b^d)$, where d is the maximum depth of the search space.

- With a good heuristic function, however, the complexity can be reduced substantially.
- The amount of the reduction depends on the particular problem and on the quality of the heuristic.

A* search

- It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost, to get from the node to the goal
- $f(n) = g(n) + h(n)$
- g(n) gives the path cost from the start node to node n,
- h(n) is the estimated cost of the cheapest path from n to the goal,
- f(n) = estimated cost of the cheapest solution through n
- A* search is both complete and optimal.

- The algorithm is identical to uniform cost search except that A* uses g + h instead of g.



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Values of h_{SLD}—straight-line distances to Bucharest.

- A* is optimally efficient for any given consistent heuristic.
- That is, no other optimal algorithm is guaranteed to expand fewer nodes than A*
- This is because any algorithm that does not expand all nodes with $f(n) < C^*$ runs the risk of missing the optimal solution.
- So A* search is complete, optimal, and optimally efficient among all such algorithms.

- But unfortunately, it does not mean that A* is the answer to all our searching needs.

Iterative Deepening A* (IDA*)

- The main difference between IDA* and standard iterative deepening is that the cutoff used is the f-cost (g+h) rather than the depth;
- at each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration.
- IDA* is practical for many problems with unit step costs and avoids the substantial overhead associated with keeping a sorted queue of nodes.

Recursive best-first search (RBFS)

- Recursive best-first search (RBFS) is a simple recursive algorithm that attempts to mimic the operation of standard best-first search, but using only linear space.
- Its structure is similar to that of a recursive depth-first search,
- But continuing indefinitely down the current path, it uses the f limit variable to keep track of the f-value of the best alternative path available from any ancestor of the current node.
- If the current node exceeds this limit, the recursion unwinds back to the alternative path.
- As the recursion unwinds, RBFS replaces the f-value of each node along the path with a backed-up value - the best f-value of its children.

- In this way, RBFS remembers the f-value of the best leaf in the forgotten subtree and can therefore decide whether it's worth reexpanding the subtree at some later time.
- RBFS is somewhat more efficient than IDA*, but still suffers from excessive node regeneration.
- IDA* and RBFS suffer from using too little memory.
- Therefore it seems sensible to use all available memory.

Simplified Memory-bounded A* (SMA*)

- SMA* proceeds just like A*, expanding the best leaf until memory is full.
- At this point, it cannot add a new node to the search tree without dropping an old one.
- SMA* always drops the worst leaf node - the one with the highest f-value.
- Like RBFS, SMA* then backs up the value of the forgotten node to its parent.
- In this way, the ancestor of a forgotten subtree knows the quality of the best path in that subtree.
- With this information, SMA* regenerates the subtree only when all other paths have been shown to look worse than the path it has forgotten.

- if all the descendants of a node n are forgotten, then we will not know which way to go from n, but we will still have an idea of how worthwhile it is to go anywhere from n.
- SMA* expands the best leaf and deletes the worst leaf.
- What if all the leaf nodes have the same f-value?
- To avoid selecting the same node for deletion and expansion, SMA* expands the newest best leaf and deletes the oldest worst leaf.
- These coincide when there is only one leaf, but in that case, the current search tree must be a single path from root to leaf that fills all of memory.
- If the leaf is not a goal node, then even if it is on an optimal solution path, that solution is not reachable with the available memory.
- Therefore, the node can be discarded exactly as if it had no successors.

Logical agents

- Humans know things; and what they know helps them do things
- They make strong claims about how the intelligence of humans is achieved
- processes of reasoning that operate on internal representations of knowledge.
- In AI, this approach to intelligence is embodied in knowledge-based agents.
- we develop logic as a general class of representations to support knowledge-based agents.
- Such agents can combine and recombine information to suit many purposes

Knowledge Base Agents

- The central component of a knowledge-based agent is its knowledge base, or KB.
- A knowledge base is a set of sentences.
- Each sentence is expressed in a language called a knowledge representation language
- Each sentence represents some assertion about the world.
- There must be a way to add new sentences to the knowledge base and a way to query what is known
- The standard names for these operations are TELL and ASK, respectively.
- Both operations may involve inference - that is, deriving new sentences from old.

- Inference must obey the requirement that when one ASKs a question of the knowledge base, the answer should follow from what has been told to the knowledge base previously.
- Outline of a knowledge-based agent program.

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))
  $action \leftarrow$ ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE($action$, t))
  $t \leftarrow t + 1$
  return *action*

- Like all our agents, it takes a percept as input and returns an action.
- The agent maintains a knowledge base, KB, which may initially contain some background knowledge
- Each time the agent program is called, it does three things.

- First, it TELLs the knowledge base what it perceives
- Second, it ASKs the knowledge base what action it should perform
- In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences, and so on
- Third, the agent program TELLs the knowledge base which action was chosen, and the agent executes the action

Wumpus world problem

- Performance measure
  - $+1000$ for climbing out of the cave with the gold,
  - –1000 for falling into a pit or being eaten by the wumpus,
  - –1 for each action taken
  - –10 for using up the arrow.
  - The game ends either when the agent dies or when the agent climbs out of the cave.
- Environment
  - A 4×4 grid of rooms.
  - The agent always starts in the square labeled $[1, 1]$, facing to the right.
  - The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square.
  - Each square other than the start can be a pit, with probability 0.2.

- Actuators
  - The agent can move Forward, TurnLeft by 90, or TurnRight by 90.
  - The agent dies a miserable death if it enters a square containing a pit or a live wumpus
  - Grab can be used to pick up the gold if it is in the same square as the agent.
  - The action Shoot can be used to fire an arrow in a straight line in the direction the agent is facing.
  - The arrow continues until it either hits the wumpus or hits a wall.
  - The agent has only one arrow, so only the first Shoot action has any effect.
  - Finally, the action Climb can be used to climb out of the cave, but only from square $[1, 1]$
- Sensors
  - In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a Stench

- In the squares directly adjacent to a pit, the agent will perceive a Breeze.
- In the square where the gold is, the agent will perceive a Glitter.
- When an agent walks into a wall, it will perceive a Bump
- When the wumpus is killed, it emits a woeful Scream that can be perceived anywhere in the cave

- The percepts will be given to the agent program in the form of a list of five symbols;
- For example, if there is a stench and a breeze, but no glitter, bump, or scream, the agent program will get [*Stench*, *Breeze*, *None*, *None*, *None*].
- Discovering the locations of pits and wumpus completes the agent's knowledge of the transition model.
- For an agent in the environment, the main challenge is its initial ignorance of the configuration of the environment;
- overcoming this ignorance seems to require logical reasoning

- Note that in each case for which the agent draws a conclusion from the available information, that conclusion is guaranteed to be correct if the available information is correct.
- This is a fundamental property of logical reasoning

Logic

- Knowledge bases consist of sentences.
- Sentences are expressed according to the syntax of the representation language
- All the sentences should be well formed
- A logic must also define the semantics or meaning of sentences.
- The semantics defines the truth of each sentence with respect to each possible world/ Model
- In standard logics, every sentence must be either true or false in each possible world/ Model
- Possible worlds might be thought of as (potentially) real environments that the agent might or might not be in,
- Models are mathematical abstractions, each of which simply fixes the truth or falsehood of every relevant sentence

- If a sentence α is true in model m, we say that m satisfies α or sometimes m is a model of α.
- We use the notation $M(\alpha)$ to mean the set of all models of α.
- A sentence follows logically from another sentence

  $\alpha \models \beta$

  mean that the sentence α entails the sentence β
- The formal definition of entailment is: $\alpha \models \beta$ if and only if, in every model in which α is true, β is also true.
- We can say $\alpha \models \beta$ if and only if $M(\alpha) \subseteq M(\beta)$
- α is a stronger assertion than β.
- The KB can be thought of as a set of sentences or as a single sentence that asserts all the individual sentences.
- The KB is false in models that contradict what the agent knows

- if KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world.

Truth Table for Logical connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
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*

Standard Logical Equivalences

- **Commutativity**
  $P \wedge Q \equiv Q \wedge P$
  $P \vee Q \equiv Q \vee P$
- **Associativity**
  $(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$
  $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$
- **Double-negation elimination**
  $\neg(\neg P) \equiv P$
- **Contraposition**
  $P \Rightarrow Q \equiv \neg Q \Rightarrow \neg P$
- **Implication elimination**
  $P \Rightarrow Q \equiv \neg P \vee Q$
- **Biconditional elimination**
  $P \Leftrightarrow Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$

- **De Morgan**
  $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$
  $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$
- **Distributivity**
  $(P \wedge Q) \vee R \equiv (P \vee R) \wedge (Q \vee R)$
  $(P \vee Q) \wedge R \equiv (P \wedge R) \vee (Q \wedge R)$
- **Modus Ponens**
  $\frac{P \Rightarrow Q, P}{Q}$
- **Modus Tollens**
  $\frac{P \Rightarrow Q, \neg Q}{\neg P}$
- **And-Elimination**
  $\frac{P \wedge Q}{P}$

Conversion to Clause Form

Algorithm to convert into clause form

1. Eliminate $\Rightarrow$ using $P \Rightarrow Q \equiv \neg P \vee Q$
2. Reduce the scope of each $\neg$ to a single term using Double-negation elimination, De Morgan's Law and $\neg\forall x : P(x) \equiv \exists x : \neg P(x)$ and $\neg\exists x : P(x) \equiv \forall x : \neg P(x)$
3. Standardize variables so that quantifier binds a unique variables.
   $\forall x : P(x) \vee \forall x : Q(x) \equiv \forall x : P(x) \vee \forall y : Q(y)$
4. Shift all quantifiers to the left of the formula without changing their relative order.
   $\forall x : P(x) \vee \forall y : Q(y) \equiv \forall x : \forall y : P(x) \vee Q(y)$

5. Eliminate existential quantifiers with particular function value. For example
   $\exists x : P(x) \equiv P(S)$ where S is a function without arguments that produces a value which satisfies P and called *Skolem Constants*.
   $\forall x : \exists y : father - of(y, x) \equiv \forall x : father - of(S(x), x)$ where $S(x)$ is a function with arguments that produces a value which satisfies $father - of$ and called *Skolem Function*.
6. Drop the prefix. Now all remaining variables are universally quantified, so the prefix just can be dropped.
7. Convert the all the clauses into conjunction of disjuncts (CNF).
8. Create separate clause corresponding to each conjunct.
9. Standardize the variables for every generated clauses. Use different variable name for each clause.

Resolution in Propositional Logic

Algorithm:

1. Convert all the propositions of F into clause from.
2. Negate P and convert the result in to clause form. Add it into the set of clauses obtained in Step 1.
3. Repeat until either a contradiction is found or no progress can be made:
   - 3.1 Select two clauses
   - 3.2 Resolve them together
   - 3.3 If the resolvent is the empty clause then a contradiction is found otherwise add it to the set of clauses available to the procedure.

Unification Algorithm

Algorithm:**Unify(L1,L2)**

1. If L1 or L2 are both variables or constants, then:
   - 1.1 if L1 and L2 are identical, then return NIL
   - 1.2 else if L1 is a variable and if L1 occurs in L2 then return FAIL else return (L2/L1)
   - 1.3 else if L2 is a variable and if L2 occurs in L1 then return FAIL else return (L1/L2)
   - 1.4 else return FAIL.
2. If the initial predicate symbols in L1 and L2 are not identical then return FAIL.
3. If L1 and L2 have a different number of arguments then return FAIL.
4. Set SUBST=NIL
5. for i=1 to number of arguments
   - 5.1 S=Unify(L1(i), L2(i))

5.2 If $FAIL \in S$ the return FAIL.
5.3 If $S \neq \emptyset$ then
5.3.1 Apply S to the remainder of both L1 and L2
5.3.2 SUBST=APPEND(S,SUBST)

6. return SUBST

Expert System

- What is an Expert System?
- Why should we use Expert System?
- History of Expert System.
- Latest Development in Expert System.
- Components of an Expert System.
- Rule Based Reasoning in Expert System.
- Limitation of Expert System.

What is an Expert System?

- An Expert System is a computer program that emulate the human expert.
- An Expert System is a software that attempts to reproduce the performance of one or more human experts, typically in a specific problem domain
- Expert System employs human knowledge represented in a computer to solve problems that ordinarily require human expertise
- Expert System imitate the expert's reasoning process to solve specific problems.

Human Expert System

Human experts have:

- A considerable knowledge about their areas of expertise
- Can learn from their experience
- Can do reasoning
- Can explain solutions
- Can restructure knowledge
- Can determine relevance

Why should we use Expert System?

- Capture and preserve irreplaceable human expertise
- Provide expertise needed at a number of locations at the same time or in a hostile environment that is dangerous to human health
- Provide unemotional objective solutions faster than human experts
- Provide expertise that is expensive or rare
- Share human expertise with a large number of people

History of Expert System

- In 1959 Newell and Simon described General Problem Solver (GPS).
  - Intended to solve general problems across domains such as theorem proving, geometric problem and chess playing
  - First computer program to separate knowledge (Set of rules and facts) from strategy (procedure)
  - Predecessor to Expert System
- Expert System introduced by Stanford Heuristic Programming Project led by Feigenbaum in mid 1960 early Expert Systems (MYCIN):
  - Diagnose infectious diseases such as bacteremia and meningitis.
  - Recommend antibiotics
  - Dosage adjusted for patient's body weight
  - Name derived from antibiotics (suffix -"mycin")
- Mid-1970s:

- Recognition of the role of knowledge
- Power of an Expert System comes from the specific knowledge it possesses, not from the inference scheme it employs
- Development of knowledge representation theories
- Development of decision making procedures and inference

- Early 1980s: Expert systems proliferated such as XCON, XSEL
  - XCON:
    - XCON (eXpert CONfigurator)
    - Used to assist in ordering of DEC's VAX computer system
    - Automatically selected the computer system components based on the customer's requirements
    - Saved DEC $25M a year by speeding the assembly process and increasing customer satisfaction
  - XSEL:
    - A newer version of XCON
    - Intended to be used by DEC's sales force
  - Universities offered Expert System courses
  - Expert System technology became commercial
  - Programming tools and shell appeared like EMYCIN, EXPERT

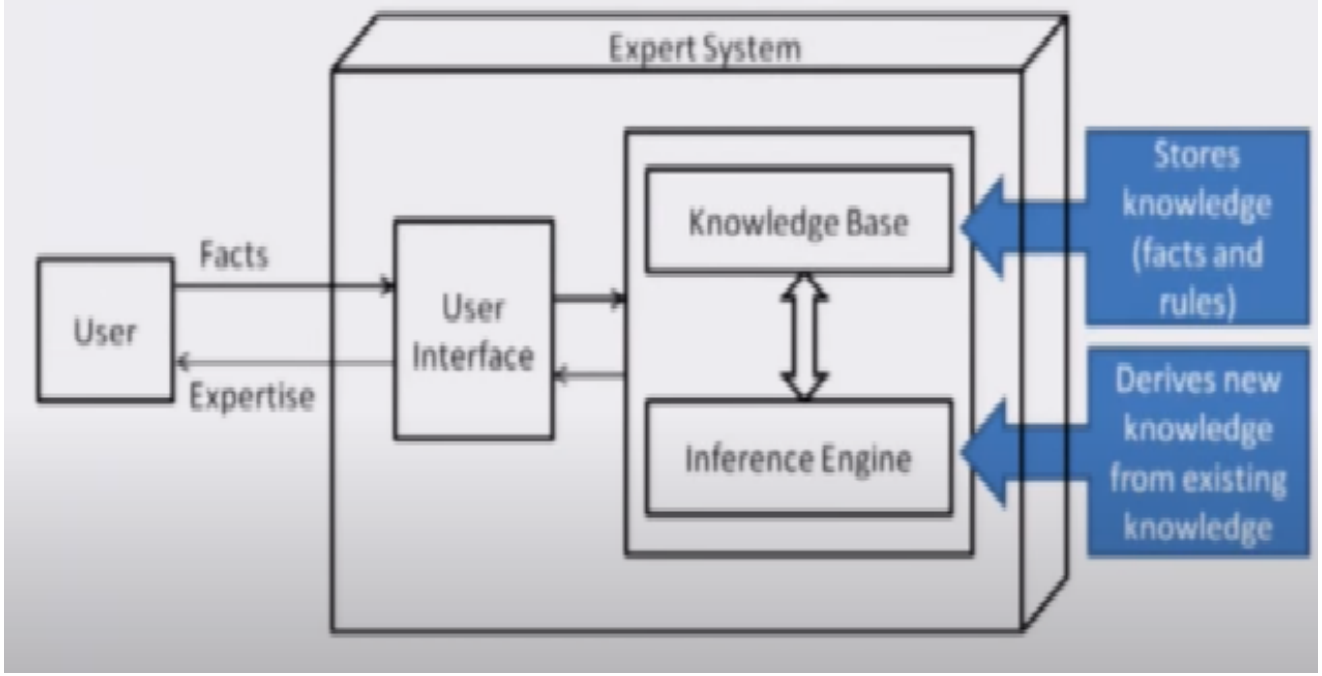
Latest Development in Expert System

- Improvements in knowledge acquisition
- Software and expertise also available on internet
- Use of multiple knowledge bases
- Many tools to expedite the construction of expert system at a reduced cost
- Expertise2Go provides a free expert system shell that provides the inference engines, acquisition and explanation interface
- Increased use of of expert systems in many tasks like medical diagnosis, car diagnosis and financial advice.

Components of an Expert System

Three main components:

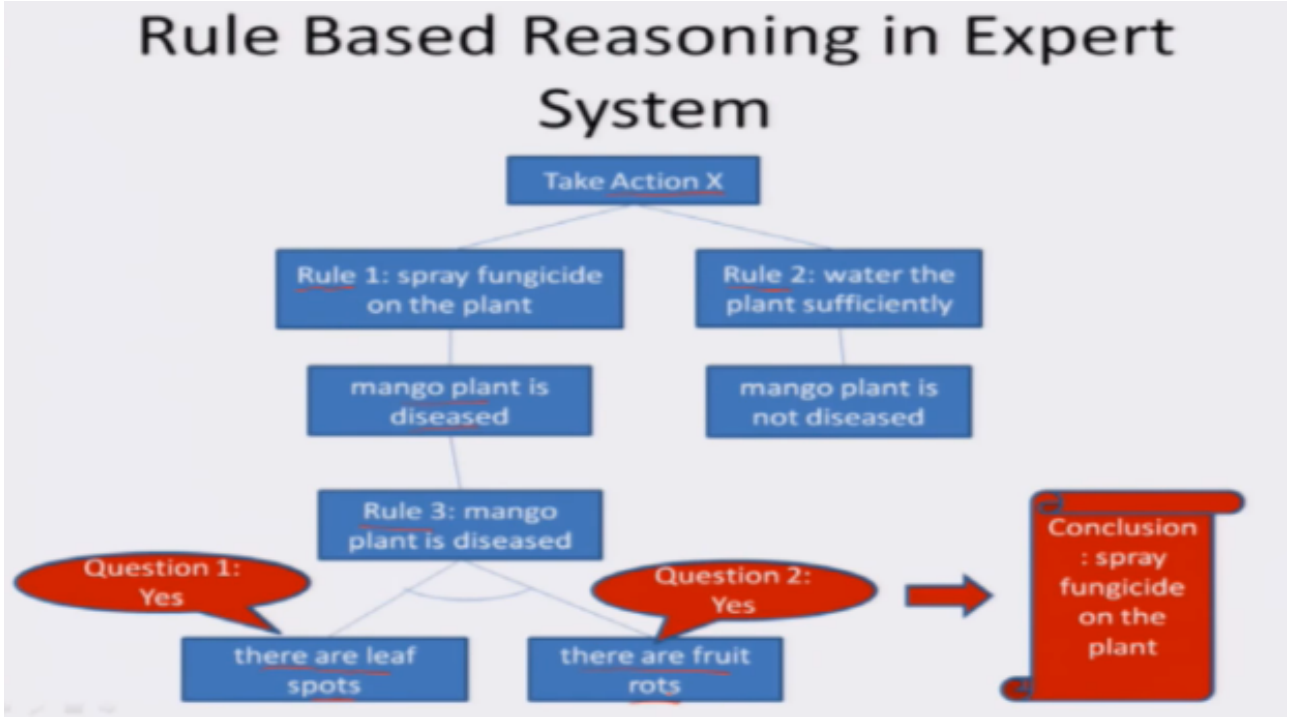
- User Interface
- Knowledge Base
- Inference Engine



Rule Based Reasoning in Expert System

Simple Rules regarding vegetation

- Rule 1:
  IF mango plat is diseased
  THEN spray fungicide on the plant
- Rule 2:
  IF mango plant is not diseased
  THEN water the plant sufficiently
- Rule 3:
  IF there are leaf spots
  AND there are fruits rots
  THEN mango plant is diseased


Rule Based Reasoning in Expert System
Take Action X
Rule 1: spray fungicide on the plant
Rule 2: water the plant sufficiently
mango plant is diseased
mango plant is not diseased
Rule 3: mango plant is diseased
Question 1: Yes
Question 2: Yes
there are leaf spots
there are fruit rots
Conclusion : spray fungicide on the plant

Explanation in Reasoning

Expert system answers two question:

- Why: means "Why are you asking for a particular information?"
  - Expert system returns the current rule that is being fired
- How? means "How did you reach the conclusion?"
  - Expert system returns the sequence of rules that are fired to reach a conclusion.

Limitation of Expert System

- Limited to relatively narrow problems
- Can't readily deal with mixed knowledge
- Errors may occur in knowledge base
- Can't refine own knowledge base or learn from experience
- Lack of common sense
- Can't make creative responses as human expert
- Domain experts not always able to explain their logic and reasoning

Learning

- What is Learning?
  - The acquisition of knowledge or skills through study, experience, or being taught.
  - Learning is a process that improves the knowledge of an AI program by making observations about its environment.
  - Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively next time.
- How do we learn.
  - Rote Learning
  - Learning by Taking Advice
  - Learning by Problem Solving
    1. Learning by Parameter Adjustment
    2. Learning by Macro Operations
    3. Learning by Chunking

- Learning by examples
- Explanation based Learning

Thank you

Please send your feedback or any queries to akyadav1@amity.edu, akyadav@akyadav.in or contact me on +91 9911375598