Probabilistic reasoning in Artificial intelligence

Uncertainty:

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

Causes of uncertainty:

Following are some leading causes of uncertainty to occur in the real world.

1. Information occurred from unreliable sources.
2. Experimental Errors
3. Equipment fault
4. Temperature variation
5. Climate change.

Probabilistic reasoning:

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.

In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- **Bayes' rule**

- **Bayesian Statistics**

As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms:

Probability: Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

1. $0 \leq P(A) \leq 1$, where P(A) is the probability of an event A.
1. $P(A) = 0$, indicates total uncertainty in an event A.
1. $P(A) = 1$, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

$$\textbf{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- $P(\neg A)$ = probability of a not happening event. ○ $P(\neg A) + P(A) = 1$.

Event: Each possible outcome of a variable is called an event.

Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects in the real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional probability:

Conditional probability is a probability of occurring an event when another event has already happened.

Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

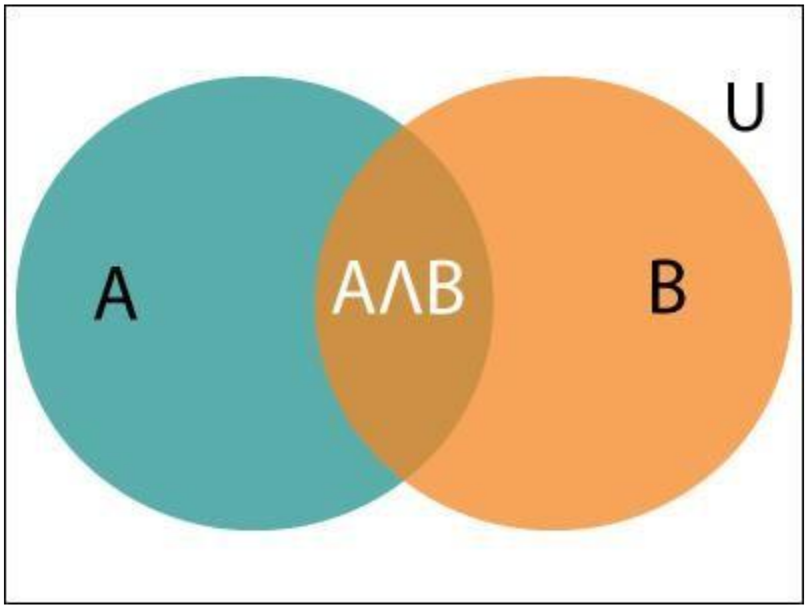
Where P($A \wedge B$)= Joint probability of a and B

P(B)= Marginal probability of B.

If the probability of A is given and we need to find the probability of B, then it will be given as:

$$P(B|A) = \frac{P(A \wedge B)}{P(A)}$$

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of **P(A∧*B*) by P(B)**.



Example:

In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?

Solution:

Let, A is an event that a student likes Mathematics B is an event that a student likes English.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

Hence, 57% are the students who like English also like Mathematics.

Bayes' theorem in Artificial intelligence

Bayes' theorem:

Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.

In probability theory, it relates the conditional probability and marginal probabilities of two random events.
Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).

Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

Example: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.

Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:

As from product rule we can write:

$$P(A \land B)= P(A|B)\ P(B) \text{ or}$$

Similarly, the probability of event B with known event A:

$$P(A \land B)= P(B|A)\ P(A)$$

Equating right hand side of both the equations, we will get:

$$\boxed{P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}} \quad ....(a)$$

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

It shows the simple relationship between joint and conditional probabilities. Here,

P(A|B) is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.

P(A) is called the **prior probability**, probability of hypothesis before considering the evidence

P(B) is called **marginal probability**, pure probability of an evidence.

Applying Bayes' rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A). This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one. Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$\mathbf{P(cause\,|\,effect)} = \frac{P(effect|cause)\ P(cause)}{P(effect)}$$

Example-1:

Question: what is the probability that a patient has diseases meningitis with a stiff neck?

Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

- The Known probability that a patient has meningitis disease is 1/30,000.
- The Known probability that a patient has a stiff neck is 2%.

Let a be the proposition that patient has stiff neck and b be the proposition that patient has meningitis. , so we can calculate the following as:

P(a|b) = 0.8

P(b) = 1/30000

P(a)= .02

$$\mathbf{P(b\,|\,a)} = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8*(\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

Example-2:

Question: From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

Solution:

$$\mathbf{P(king|face)} = \frac{P(Face|king) * P(King)}{P(Face)} \quad ....(i)$$

P(king): probability that the card is King= 4/52= 1/13

P(face): probability that a card is a face card= 3/13

P(Face|King): probability of face card when we assume it is a king = 1

Substituting all values in equation (i) we will get:

$$P(king|face) = \frac{1 * (\frac{1}{13})}{(\frac{3}{13})} = \mathbf{1/3}, \text{ it is a probability that a face card is a king card.}$$

Application of Bayes' theorem in Artificial intelligence:

Following are some applications of Bayes' theorem:

- It is used to calculate the next step of the robot when the already executed step is given.
- Bayes' theorem is helpful in weather forecasting.
- It can solve the Monty Hall problem.

Bayesian Belief Network in artificial intelligence

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.

Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

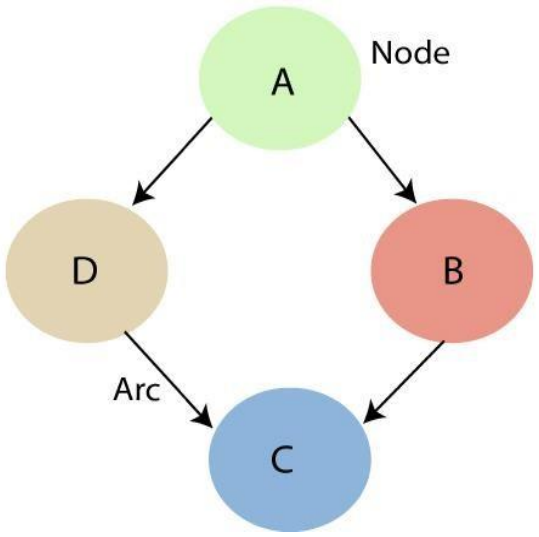
Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- **Directed Acyclic Graph**
- **Table of conditional probabilities.**

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



- Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.
- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph. These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
- **In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
- **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
- **Node C is independent of node A.**

Note: The Bayesian network graph does not contain any cyclic graph. Hence, it is known as a directed acyclic graph or DAG**.**

The Bayesian network has mainly two components:

- **Causal Component**
- **Actual numbers**

Each node in the Bayesian network has condition probability distribution **P(X_i |Parent(X_i))**, which determines the effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3,....., xn, then the probabilities of a different combination of x1, x2, x3.. xn, are known as Joint probability distribution.
P[x_1, x_2, x_3,....., x_n], it can be written as the following way in terms of the joint probability distribution.

= P[x_1| x_2, x_3,....., x_n]P[x_2, x_3,....., x_n]

= P[x_1| x_2, x_3,....., x_n]P[x_2|x_3,....., x_n]....P[x_{n-1}|x_n]P[x_n].

In general for each variable Xi, we can write the equation as:

$P(X_i|X_{i-1},........., X_1) = P(X_i |Parents(X_i))$

Explanation of Bayesian network:

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

Example: Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

Problem:

Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

Solution:

- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table or CPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a boolean variable with k boolean parents contains 2^K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

List of all events occurring in this network:

- **Burglary (B)** ○ **Earthquake(E)** ○ **Alarm(A)**
- **David Calls(D)** ○ **Sophia calls(S)**

We can write the events of problem statement in the form of probability: **P[D, S, A, B, E]**, can rewrite the above probability statement using joint probability distribution:

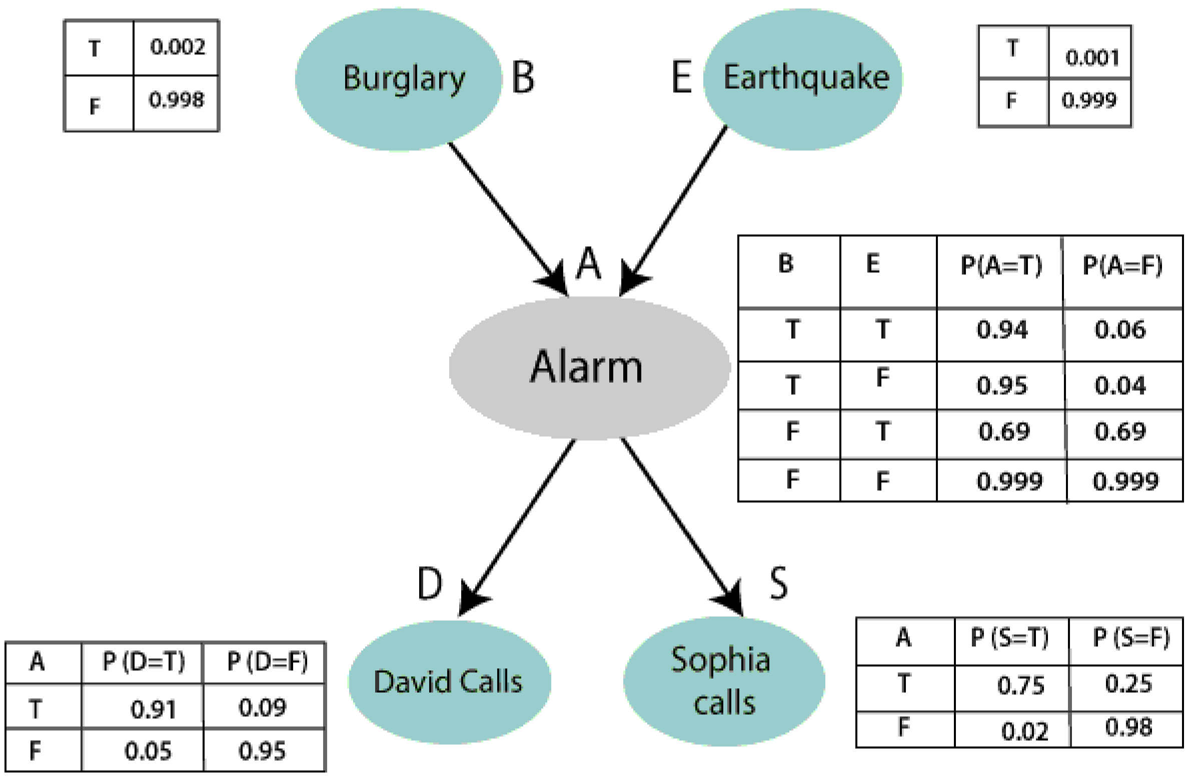
P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]

=P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E]

= P [D| A]. P [S| A, B, E]. P[A, B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B |E]. P[E]



T	0.002
F	0.998

T	0.001
F	0.999

B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P (D=T)	P (D=F)
T	0.91	0.09
F	0.05	0.95

A	P (S=T)	P (S=F)
T	0.75	0.25
F	0.02	0.98

Let's take the observed probability for the Burglary and earthquake component:

P(B= True) = 0.002, which is the probability of burglary.

P(B= False)= 0.998, which is the probability of no burglary.

P(E= True)= 0.001, which is the probability of a minor earthquake

P(E= False)= 0.999, Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

Conditional probability table for Alarm A:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	**E**	**P(A= True)**	**P(A= False)**
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	**P(D= True)**	**P(D= False)**
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	**P(S= True)**	**P(S= False)**
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E).

= 0.75* 0.91* 0.001* 0.998*0.999

= 0.00068045.

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

The semantics of Bayesian Network:

There are two ways to understand the semantics of the Bayesian network, which is given below:

1. To understand the network as the representation of the Joint probability distribution.

It is helpful to understand how to construct the network.

2. To understand the network as an encoding of a collection of conditional independence statements.

DEMPSTER-SHAFER THEORY

The theory of belief functions, also referred to as evidence theory or Dempster– Shafer theory (DST), is a general framework for reasoning with uncertainty, with understood connections to other frameworks such as probability, possibility and imprecise probability theories. First introduced by Arthur P. Dempster in the context of statistical inference, the theory was later developed by Glenn Shafer into a general framework for modeling epistemic uncertainty—a mathematical theory of evidence. The theory allows one to combine evidence from different sources and arrive at a degree of belief (represented by a mathematical object called belief function) that takes into account all the available evidence. The Dempster-Shafer theory, also known as the theory of belief functions, is a generalization of the Bayesian theory of subjective probability.

Whereas the Bayesian theory requires probabilities for each question of interest, belief functions allow us to base degrees of belief for one question on probabilities for a related question. These degrees of belief may or may not have the mathematical properties of probabilities;The Dempster-Shafer theory owes its name to work by A. P. Dempster

(1968) and Glenn Shafer (1976), but the theory came to the attention of AI researchers in the early 1980s, when they were trying to adapt probability theory to expert systems.Dempster-Shafer degrees of belief resemble the certainty factors in MYCIN, and this resemblance suggested that they might combine the rigor of probability theory with the flexibility of rule-based systems.

The Dempster-Shafer theory remains attractive because of its relative flexibility. The Dempster-Shafer theory is based on two ideas:the idea of obtaining degrees of belief for one question from subjective probabilities for a related question,Dempster's rule for combining such degrees of belief when they are based on independent items of evidence.

In a narrow sense, the term Dempster–Shafer theory refers to the original conception of the theory by Dempster and Shafer. However, it is more common to use the term in the wider sense of the same general approach, as adapted to specific kinds of situations. In particular, many authors have proposed different rules for combining evidence, often with a view to handling conflicts in evidence better. The early contributions have also been the starting points of many important developments, including the transferable belief model and the theory of hints.

Dempster–Shafer theory is a generalization of the Bayesian theory of subjective probability. Belief functions base degrees of belief (or confidence, or trust) for one question on the probabilities for a related question. The degrees of belief themselves may or may not have the mathematical properties of probabilities; how much they differ depends on how closely the two questions are related. Put another way, it is a way of representing epistemic plausibilities but it can yield answers that contradict those arrived at using probability theory.

Often used as a method of sensor fusion, Dempster–Shafer theory is based on two ideas: obtaining degrees of belief for one question from subjective probabilities for a related question, and Dempster's rule for combining such degrees of belief when they are based on independent items of evidence. In essence, the degree of belief in a proposition depends primarily upon the number of answers (to the related questions) containing the proposition, and the subjective probability of each answer. Also contributing are the rules of combination that reflect general assumptions about the data.In this formalism a degree of belief (also referred to as a mass) is represented as a belief function rather than a Bayesian probability

distribution. Probability values are assigned to sets of possibilities rather than single events: their appeal rests on the fact they naturally encode evidence in favor of propositions.

Dempster–Shafer theory assigns its masses to all of the subsets of the propositions that compose a system—in set-theoretic terms, the power set of the propositions. For instance, assume a situation where there are two related questions, or propositions, in a system. In this system, any belief function assigns mass to the first proposition, the second, both or neither.

Planning

Assumptions are:
(1) Environment is deterministic
(2) Environment is observable
(3) Environment is static (it only in response to the agent’s actions)

- State of the world is represented as collection of variables-Planning Domain Definition Language:PDDL
- search problem: the initial state, the actions that are available in a state, the result of applying an action, and the goal test.
- Each state is represented as a conjunction of fluents that are ground, functionless atoms
- Example: a state in a package delivery problem might be At(Truck 1, Melbourne) ∧ At(Truck 2, Sydney).
- A set of ground (variable-free) actions can be represented by a single action schema.
- The schema is a lifted representation—it lifts the level of reasoning from propositional logic to a restricted subset of first-order logic.
- For example, here is an action schema for flying a plane from one location to another:
    - Action(Fly(p, from, to),

- PRECOND: At(p, from) ∧ Plane(p) ∧ Airport (from) ∧ Airport (to)

  (the states in which the action can be executed)

- EFFECT: ¬ At(p, from) ∧ At(p, to))

  (defines the result of executing the action)

- The schema consists of the action name, a list of all the variables used in the schema, a precondition and an effect.
- action that results from substituting values for all the variables:

  Action(Fly(P1, SFO, JFK),

  PRECOND:At(P1, SFO) ∧ Plane(P1) ∧ Airport (SFO) ∧ Airport (JFK)

  EFFECT: ¬ At(P1, SFO) ∧ At(P1, JFK))

- The precondition and effect of an action are each conjunctions of literals (positive or negated atomic sentences).
- An action a can be executed in state s if s entails the precondition of a.
- Entailment can also be expressed with the set semantics:

  s |= q iff every positive literal in q is in s and every negated literal in q is not.

- In formal notation we say

  (a ∈ ACTIONS(s)) ⇔ s |= PRECOND(a) ,

- where any variables in a are universally quantified.

For example,

∀ p, from, to (Fly(p, from, to) ∈ ACTIONS(s)) ⇔

s |= (At(p, from) ∧ Plane(p) ∧ Airport (from) ∧ Airport (to))

- We say that action a is applicable in state s if the preconditions are satisfied by s. When an action schema a contains variables, it may have multiple applicable instance
- Fly action can be instantiated as Fly(P1, SFO, JFK) or as Fly(P2, JFK, SFO), both of which are applicable in the initial state.

- If an action a has v variables, then, in a domain with k unique names of objects, it takes $O(v^k)$ time in the worst case to find the applicable ground actions.
- The result of executing action a in state s is defined as a state s which is represented by the set of fluents formed by starting with s, removing the fluents that appear as negative
- DELETE LIST literals in the action's effects (what we call the delete list or DEL(a)), and adding the fluents that are positive literals in the action's effects (what we call the add list or ADD(a)):

  RESULT(s, a) = (s − DEL(a))∪ ADD(a)
- For example, with the action Fly(P1, SFO, JFK), we would remove At(P1, SFO) and add At(P1, JFK).
- It is a requirement of action schemas that any variable in the effect must also appear in the precondition.
- That way, when the precondition is matched against the state s, all the variables will be bound, and RESULT(s, a) will therefore have only ground atoms.

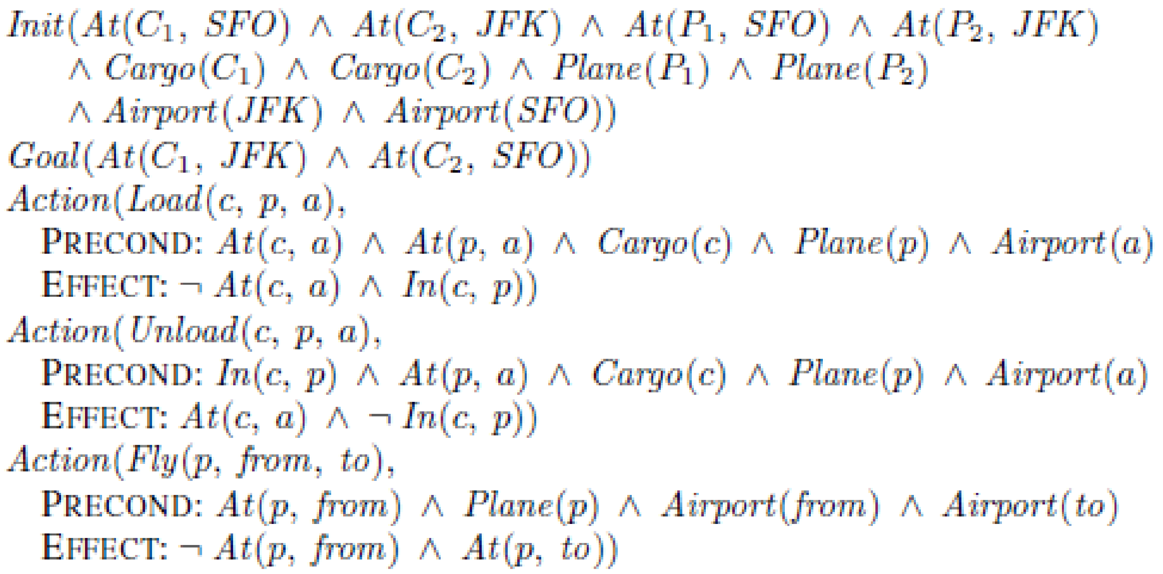
  Example: Air cargo transport

- three actions: Load , Unload, and Fly
- The actions affect two predicates: In(c, p) means that cargo c is inside plane p,

  and At(x, a) means that object x (either plane or cargo) is at airport a.
- Note that some care must be taken to make sure the At predicates are maintained properly.
- The following plan is a solution to the problem:

  [Load (C1, P1, SFO), Fly(P1, SFO, JFK),Unload(C1, P1, JFK),

  Load (C2, P2, JFK), Fly(P2, JFK, SFO),Unload(C2, P2, SFO)] .



$Init(At(C_1,\ SFO) \land At(C_2,\ JFK) \land At(P_1,\ SFO) \land At(P_2,\ JFK)$
$\quad \land\ Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\quad \land\ Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1,\ JFK) \land At(C_2,\ SFO))$
$Action(Load(c,\ p,\ a),$
$\quad$ PRECOND: $At(c,\ a) \land At(p,\ a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $\neg At(c,\ a) \land In(c,\ p))$
$Action(Unload(c,\ p,\ a),$
$\quad$ PRECOND: $In(c,\ p) \land At(p,\ a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $At(c,\ a) \land \neg In(c,\ p))$
$Action(Fly(p,\ from,\ to),$
$\quad$ PRECOND: $At(p,\ from) \land Plane(p) \land Airport(from) \land Airport(to)$
$\quad$ EFFECT: $\neg At(p,\ from) \land At(p,\ to))$

Figure 10.1 A PDDL description of an air cargo transportation planning problem.

- Finally, there is the problem of spurious actions such as Fly(P1, JFK, JFK), which should be a no-op, but which has contradictory effects (according to the definition, the effect would include At(P1, JFK) ∧ ¬ At(P1, JFK)).
- It is common to ignore such problems, because they seldom cause incorrect plans to be produced.
- The correct approach is to add inequality preconditions saying that the from and to airports must be different;

Example: The blocks world

- This domain consists of a set of cube-shaped blocks sitting on a table.
- The blocks can be stacked, but only one block can fit directly on top of another.
- A robot arm can pick up a block and move it to another position, either on the table or on top of another block.
- The arm can pick up only one block at a time, so it cannot pick up a block that has another one on it.
- The goal will always be to build one or more stacks of blocks, specified in terms of what blocks are on top of what other blocks

Actions:

- UNSTACK(A,B) – Pick up block A from its current position on Block B. The arm must be empty and block A must have no blocks on top of it.
- STACK(A,B) – Place block A on Block B. The arm must already be holding and the surface of B must be clear.
- PICKUP(A)- Pick up block A from the table and hold it. The arm must be empty and there must be nothing on the top of block A.
- PUTDOWN(A)- Put block A down on the table. The arm must have been holding block A.

The robot arm can hold one block at a time. Since all blocks are of same size, each block can have atmost one other block directly on the top of it.

For the operations we need the following predicates.

- ON(A,B) – Bock A is on Block B
- ONTABLE(A)- block A is on table
- CLEAR(A)- there is nothing on the top of A
- HOLDING(A)- The arm is holding block A
- ARMEMPTY- The arm is holding nothing

Some logical statements:

$[\exists x{:}\mathrm{HOLDING}(x)] \rightarrow \neg \mathrm{ARMEMPTY}$

If the arm is holding anything then it is not empty.

$\forall x{:}\ \mathrm{ONTABLE}(x) \rightarrow \neg \exists y{:}\mathrm{ON}(x,y)$

If a block is on table, then it is not also on another block

$\forall x{:}\ [\neg \exists y{:}\ \mathrm{ON}(y,x)] \rightarrow \mathrm{CLEAR}(x)$

Any block with no block on it is clear

Components of planning system:

1.**Choosing rules to apply**

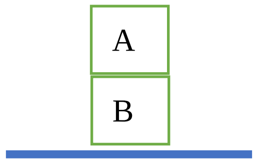
The most widely used technique for selecting rules to apply is first to isolate a set of differences between the desired goal state and the current state and then to identify those rules that are relevant to reducing those differences. If several rules are found, a variety of other heuristic information can be exploited to choose among them. Means end analysis- For example , fi our goal is to have a white fence around out yard and we currently have a brown fence, we would select operators whose result involves a change of color of an object. If, on the other hand, we currently have no fencer, we must first consider operators that involve constructing wooden objects.

2.Applying Rules:

In simple systems applying rules was easy. Each rule simply specified the problem state that would result from its application. We must be able to do with rules that specify only a small part of the complete problem state. There are many ways for doing this.

1. For each action, describe the each of changes it makes to the state description. In addition, some statements, that everything else remains unchanged is also necessary. In a system, a given state was described by set of predicates representing the facts that were true in each state. Each distinct state was represented explicitly as part of the predicate.
   Example the state S0 of sample blocks world problem could be represented.

   On(A,B,S0) Λ ONTABLE(B,S0) Λ CLEAR(A,S0)



The manipulation of these state descriptions was done using a resolution theorem prover. So, for example, the effect of the operator UNSTACK(x,y) could be described by the following axiom.

$$[CLEAR(x,s) \wedge ON(x,y,s)] \rightarrow [HOLDING(x,DO(UNSTACK(x,y),s)) \wedge CLEAR(y,DO(UNSTACK(x,y,s))]$$

Here DO is a function that specifies , for a given state and a given action, the new state that results from the execution of the action. The axiom state that if CLEAR(x) and ON(x,y) both hold in state s,

then HOLDING(x) and CLEAR(y) will hold in the state that results from Doing an UNSTACK(x,y), starting in state s.
If we execute UNSTACK(A,B) in state S0 as defined above, then we can prove , using our assertions about S0 and our axiom about UNSTACK, that in the state results from unstacking operarion (we call this state S1)

HOLDING(A,S1) Λ CLEAR(B,S1)

To handle complex problem domains, we need a mechanism that does not require a large number of explicit frame axioms. One such mechanism is that used by the early robot problem solving system STRIPS and its descendants. In these approach, each operation is described by a list of new predicates that the operator causes to become true and a list of old predicates that it causes to become false. These two lists are called ADD and DELETE lists, respectively. Any predicate not included on either the ADD or DELETE list of an operator is assumed to be unaffected by it. Thus we need say nothing about the relationship of UNSTACK to COLOR. Of course , this means some mechanism other than simple theorem proving must be used to compute complete state of descriptions after operations have been performed.

STRIPS – style operators:

STACKS(x,y)	P:CLEAR(y) Λ HOLDING(x)
	D: CLEAR(y) Λ HOLDING(x)
	A:ARMEMPTYΛ ON(x,y)
UNSTACK(x,y)	P:ON(x,y) Λ CLEAR(x) Λ ARMEMPTY
	D: ON(x,y) Λ ARMEMPTY
	A: HOLDING(x) Λ CLEAR(y)
PICKUP(x)	P: CLEAR(x) Λ ONTABLE(x) Λ ARMEMPTY
	D: ONTABLE(x) Λ ARMEMPTY
	A: HOLDING(x)
PUTDOWN(x)	P: HOLDING(x)
	D: HOLDING(x)
	A: ONTABLE(x) Λ ARMEMPTY

Notice that the simple rule such as these the PRECONDITION list is often identical to the DELETE list. In order to pick up a block , the robot arm must be empty ;as soon as it picks up a block, it is no longer empty. But preconditions are not always deleted. For example, in order for the arm to pick up a block, the block must have on the top of it. After it is picked up, it still has no blocks on the top of it. This is the reason that the PRECONDITION and DELETE lists must be specified separately.

3.Detecting a solution

Planning system is succeeded in finding a solution to the problem, when the sequence of operation that transforms the initial state to goal state is found. In simple problem we can check the completion of task by goal test. But for complex problem, it depends on the way that state descriptions are represented. The representation scheme used must have capability to reason with representations to discover whether one matches another. Predicate logic has that capacity, with detective mechanism. Assume a predicate P(x). To see whether P(x) is satisfied in some state, we ask whether we can prove P(x) given the assertions that describe the state and the axioms that define the world model (such arm holding something or not empty) If we construct such a proof then the problem-solving process terminates. If we cannot, then a sequence of operators that might solve the problem must be proposed. These sequence can then be tested in the same way as the initial state was by asking whether P(x) can be proved from the axioms and the state description that was derived by applying the operators.

4.Detecting the dead ends

Detection of dead end is important. The same reasoning mechanism can be used to find dead end. If the search process is reasoning forward from the initial state, it can prune any path that leads to a state from which the goal state cannot be reached. For example, suppose we have a fixed supply of paint: some white some pink and some red. We want to paint a room so that it has light red walls and a white ceiling. We could produce light red paint by adding some white paint to red. But then we cannot paint the ceiling with white. This approach (red+white) should be abandoned in favour of mixing the pink and red paints together. Prune the path and take the path closer to solution.
In reasoning backword, each goal is decomposed into subgoals. Each in turn leads to additional subgoals. Soemetimes it is easy to detect that there is no way that all the subgoals in a given set can be satisfied at once. For example the robot arm cannot be both empty and holding a block. Any path that is attemting to make both of these goals true simultaneously can be pruned immediately. Other paths can be pruned because that lead nowhere.

5.Repairing an almost correct solution

One good way of solving decomposable problems is to assume that they are completely decomposable and find the subproblems to be solved. Still the solution is not possible then look for another one. Slightly different approach, do sequence of some operations which leads to closer to goal state and now try to solve by subproblem. Even better approach, is having specific knowledge about what went wrong and then apply to direct patch. , such as inadequate operator can be invoked to satisfy precondition. The last one is least-commitment strategy. Instead of following order of for preconditions, look the effect of each sub solutions to determine the dependencies that exist among them . at that point , an ordering can be chosen.

Goal stack planning

One of the earlier planning algorithms called goal stack planning. It was used by STRIPS.

We work backwards from the goal, looking for an operator which has one or more of the goal literals as one of its effects and then trying to satisfy the preconditions of the operator. The preconditions of the operator become subgoals that must be satisfied. We keep doing this until we reach the initial state.

Planning is process of determining various actions that often lead to a solution.

Planning is useful for non-decomposable problems where subgoals often interact.

Goal Stack Planning (in short GSP) is the one of the simplest planning algorithm that is designed to handle problems having compound goals. And it utilizes STRIP as a formal language for specifying and manipulating the world with which it is working.

This approach uses a Stack for plan generation. The stack can contain Sub-goal and actions described using predicates. The Sub-goals can be solved one by one in any order.

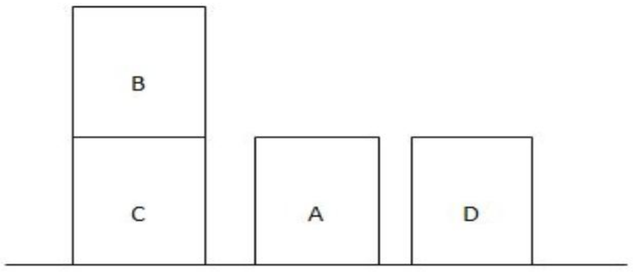
```
Push the Goal state in to the Stack
Push the individual Predicates of the Goal State into the Stack
Loop till the Stack is empty
    Pop an element E from the stack
    IF E is a Predicate
        IF E is True then
            Do Nothing
        ELSE
            Push the relevant action into the Stack
            Push the individual predicates of the Precondition of the action into the Stack
    Else IF E is an Action
        Apply the action to the current State.
        Add the action ‘a’ to the plan
```

Explanation:

The Goal Stack Planning Algorithms works will the stack. It starts by pushing the unsatisfied goals into the stack. Then it pushes the individual subgoals into the stack and its pops an element out of the stack. When popping an element out of the stack the element could be either a predicate describing a situation about our world or it could be an action that can be applied to our world under consideration. So based on the kind of element we are popping out from the stack a decision has to be made. If it is a Predicate. Then compares it with the description of the current world, if it is satisfied or is already present in our current situation then there is nothing to do because already its true.On the contrary if the Predicate is not true then we have to select and push relevant action satisfying the predicate to the Stack.

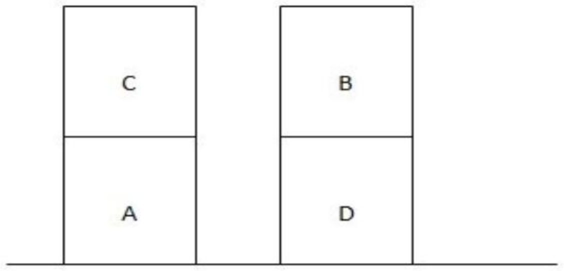
So after pushing the relevant action into the stack its precondition should also has to be pushed into the stack. In order to apply an operation its precondition has to be satisfied. In other words the present situation of the world should be suitable enough to apply an operation. For that, the preconditions are pushed into the stack once after an action is pushed.

Example:



Initial State

ON(B,C) $\wedge$ ONTABLE(C) $\wedge$ ONTABLE(A) $\wedge$ ONTABLE(D)
CLEAR(B) $\wedge$ CLEAR(A) $\wedge$ CLEAR(D)

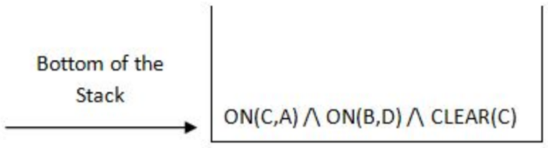


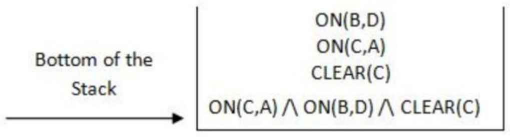
Goal State

ON(C,A) $\wedge$ ON(B,D) $\wedge$ ONTABLE(A) $\wedge$ ONTABLE(D)
$\wedge$ CLEAR(C) $\wedge$ CLEAR(B)

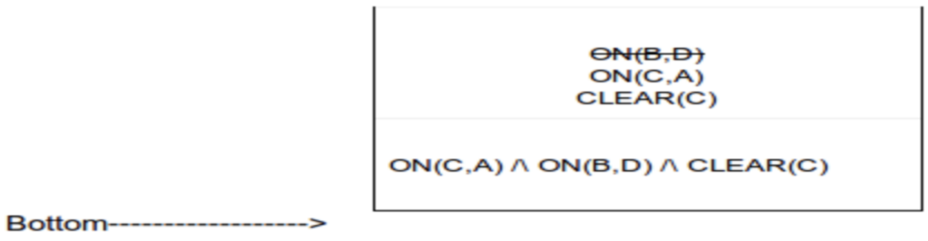
First step is to push the goal into the stack.

Next push the individual predicates of the goal into the stack.

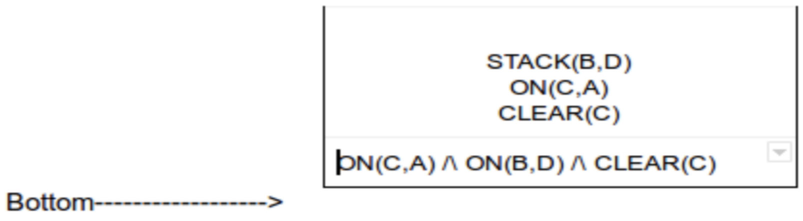




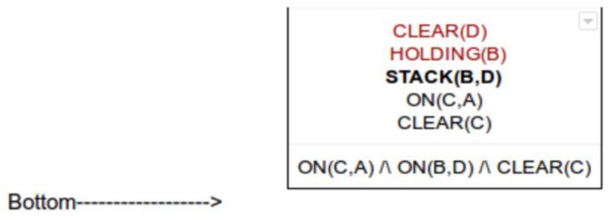
Now pop an element out from the stack



The popped element is indicated with a strike-through in the above diagram. The element is ON(B,D) which is a predicate and it is not true in our current world. So the next step is to push the relevant action which could achieve the subgoal ON(B,D) in to the stack.



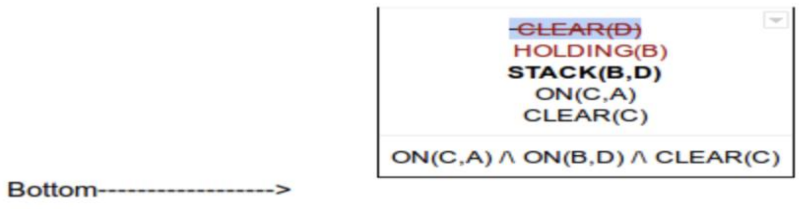
Now again push the precondition of the action Stack(B,D) into the stack.



The preconditions are highlighted with red and the actions are marked with Bold. Now coming to the point we have to make the precondition to become true in order to apply the action. [Here we need to note an interesting thing about the preconditions that, it can be pushed into the stack in any order but in certain situations we have to make some exception and that is clearly seen in the above diagram. The HOLDING(B) is pushed first and CLEAR(D) is pushed next indicating that the HOLDING subgoal has to be done second comparing with the CLEAR. Because we are considering the block world with single arm robot and everything that we usually do here is depending on the robotic arm if we first achieve HOLDING(D) then we have to undo the subgoal in order the achieve some other subgoal. So if our compound goal has HOLDING (x) subgoal, achieve it at the end.]

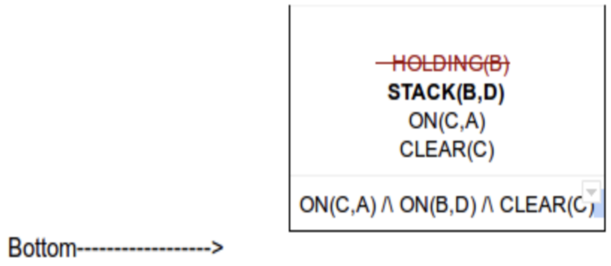
Lets go back to our example here,

POP an element out from the stack.



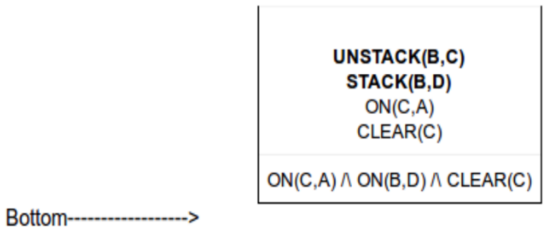
After popping we see that CLEAR(D) is true in the current world model so we don’t have to do anything.

So again pop the stack,

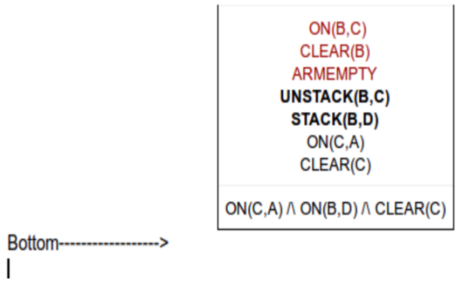


The popped element is HOLDING(B) which is a predicate and note that it is not true in our current world. So we have to push the relevant action into the stack. Inorder to make the HOLDING(D) to be true there are possibly two action that can achieve it.One is PICKUP(D) and the other is UNSTACK(D,y). But now in-order to choose the best among the two actions available we have to think ahead and utilize the heuristics possibly. For instance if we choose PICKUP(B) then first of all BLOCK D should be available on the table. For that we have to UNSTACK(B,D) and it will achieve HOLDING(B) which is what we want but if we use PICKUP then we need to PUTDOWN(B) making HOLDING(B) false and then use PICKUP(B) action to achieve HOLDING(B) again which can be easily achieved by using UNSTACK. So the best action is UNSTACK(B,y) and it also makes the current situation more close to the goal state. The variable y indicates any block below D.

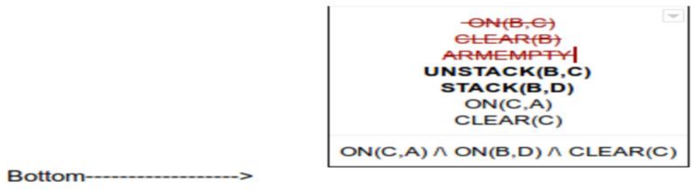
Lets push the action UNSTACK(B,C) into the stack.



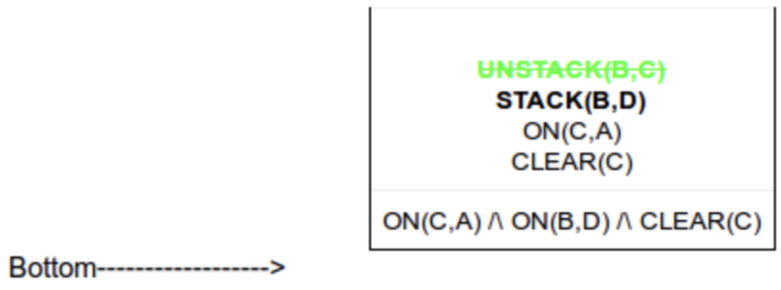
Now push the individual precondition of UNSTACK(B,C) into the stack.



POP the stack. Note here that on popping we could see that ON(B,C) ,CLEAR(B) AND ARMEMPTY are true in our current world. So dont do anything.



Now again pop the stack .

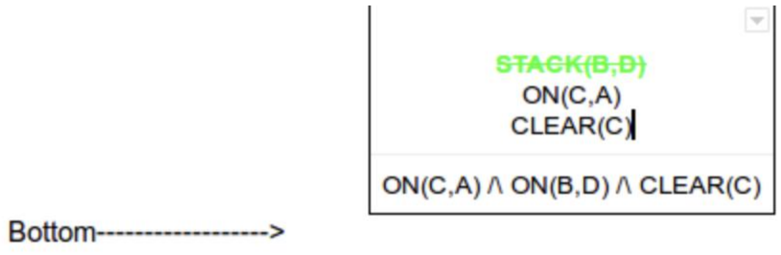


When we do that we will get an action, so just apply the action to the current world and add that action to plan list.

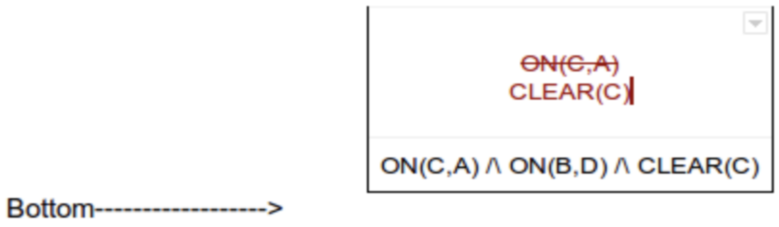
Plan= { UNSTACK(B,C) }

Again pop an element. Now its STACK(B,D) which is an action so apply that to the current state and add it to the PLAN.

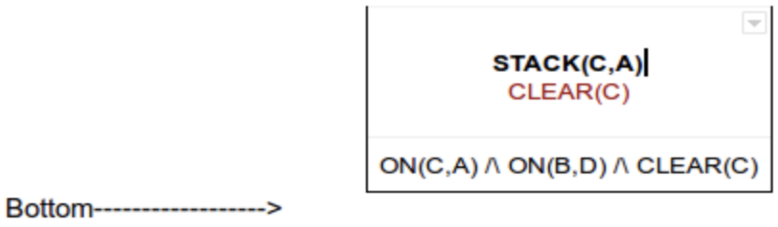
PLAN= { UNSTACK(B,C), STACK(B,D) }



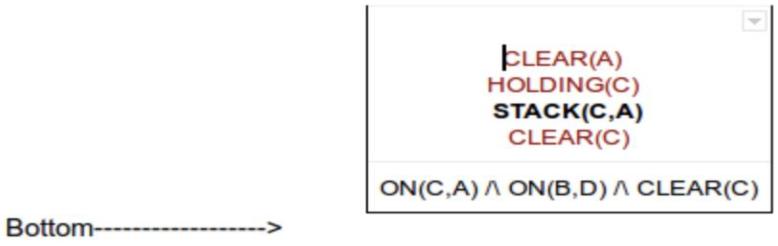
Now the stack will look like the one given below and our current world is like the one above.



Again pop the stack. The popped element is a predicate and it is not true in our current world so push the relevant action into the stack.

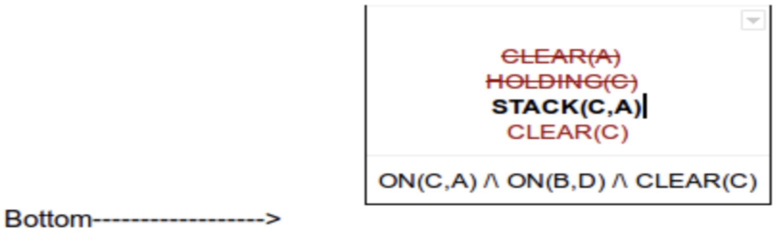


STACK(C,A) is pushed now into the stack and now push the individual preconditions of the action into the stack.



Now pop the stack. We will get CLEAR(A) and it is true in our current world so do nothing.

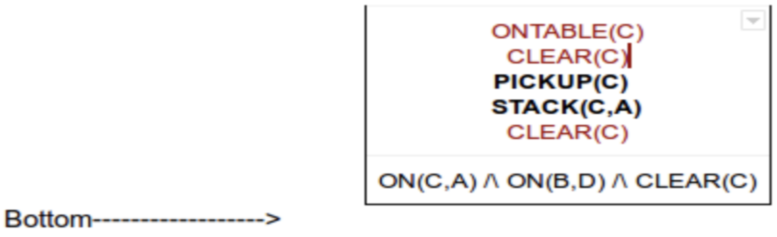
Next element that is popped is HOLDING(C) which is not true so push the relevant action into the stack.


~~CLEAR(A)~~
~~HOLDING(C)~~
STACK(C,A)
CLEAR(C)

ON(C,A) ∧ ON(B,D) ∧ CLEAR(C)

Bottom------------------->

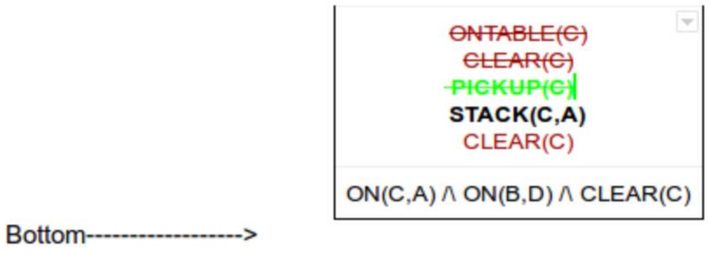
In order to achieve HOLDING(C) we have to push the action PICKUP(C) and its individual preconditions into the stack.


ONTABLE(C)
CLEAR(C)
PICKUP(C)
STACK(C,A)
CLEAR(C)

ON(C,A) ∧ ON(B,D) ∧ CLEAR(C)

Bottom------------------->

Now doing pop we will get ONTABLE(C) which is true in our current world.Next CLEAR(C) is popped and that also is achieved.Then PICKUP(C) is popped which is an action so apply it to the current world and add it to the PLAN. The world model and stack will look like below,

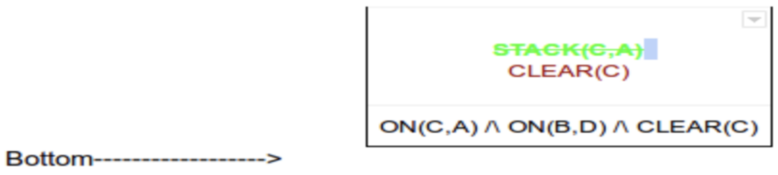

~~ONTABLE(C)~~
~~CLEAR(C)~~
~~PICKUP(C)~~
STACK(C,A)
CLEAR(C)

ON(C,A) ∧ ON(B,D) ∧ CLEAR(C)

Bottom------------------->

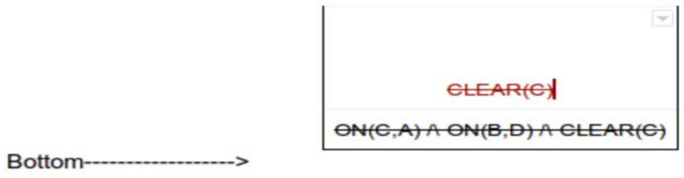
PLAN= { UNSTACK(B,C), STACK(B,D) ,PICKUP(C) }

Again POP the stack, we will get STACK(C,A) which is an action apply it to the world and insert it to the PLAN.



PLAN= { UNSTACK(B,D), STACK(B,D) ,PICKUP(C) ,STACK(C,A) }

Now pop the stack we will get CLEAR(C) which is already achieved in our current situation. So we don't need to do anything. At last when we pop the element we will get all the three subgoal which is true and our PLAN will contain all the necessary actions to achieve the goal.



PLAN= { UNSTACK(B,D), STACK(B,D) ,PICKUP(C) ,STACK(C,A) }

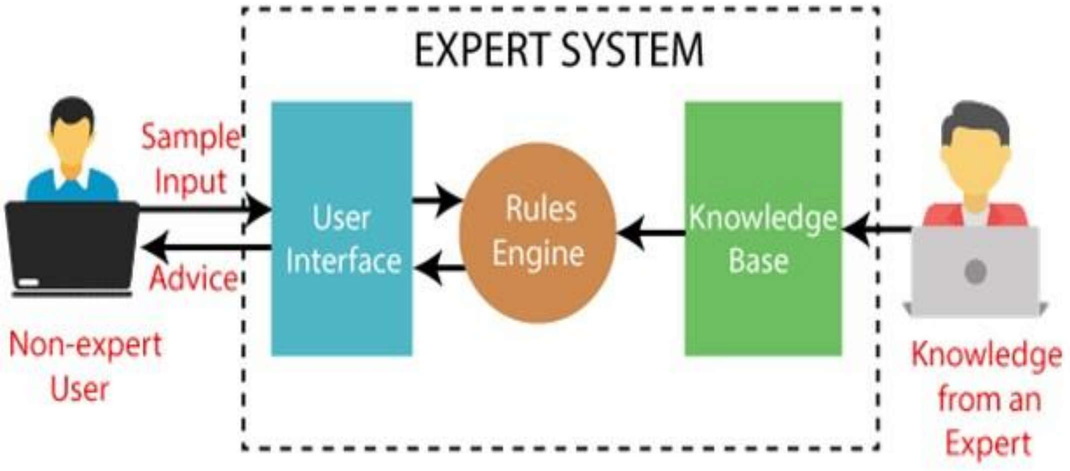
Expert System

An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

The expert system is a part of AI, and the first ES was developed in the year 1970, which was the first successful approach of artificial intelligence. It solves the most complex issue as an expert by extracting the knowledge stored in its knowledge base. The system helps in decision making for complex problems using **both facts and heuristics like a human expert**. It is called so because it contains the expert knowledge of a specific domain and can solve any complex problem of that particular domain. These systems are designed for a specific domain, such as **medicine, science,** etc.

The performance of an expert system is based on the expert's knowledge stored in its knowledge base. The more knowledge stored in the KB, the more that system improves its performance. One of the common examples of an ES is a suggestion of spelling errors while typing in the Google search box.

Below is the block diagram that represents the working of an expert system:



Note: It isimportant to remember that an expert system is not used to replace the human experts; instead, it is used to assist the human in making a complex decision. These systems do not have human capabilities of thinking and work on the basis of the knowledge base of the particular domain.

Below are some popular examples of the Expert System:

- **DENDRAL:** It was an artificial intelligence project that was made as a chemical analysis expert system. It was used in organic chemistry to detect unknown organic molecules with the help of their mass spectra and knowledge base of chemistry.

- **MYCIN:** It was one of the earliest backward chaining expert systems that was designed to find the bacteria causing infections like bacteraemia and meningitis. It was also used for the recommendation of antibiotics and the diagnosis of blood clotting diseases.
- **PXDES:** It is an expert system that is used to determine the type and level of lung cancer. To determine the disease, it takes a picture from the upper body, which looks like the shadow.
  This shadow identifies the type and degree of harm.
- **CaDeT:** The CaDet expert system is a diagnostic support system that can detect cancer at early stages.

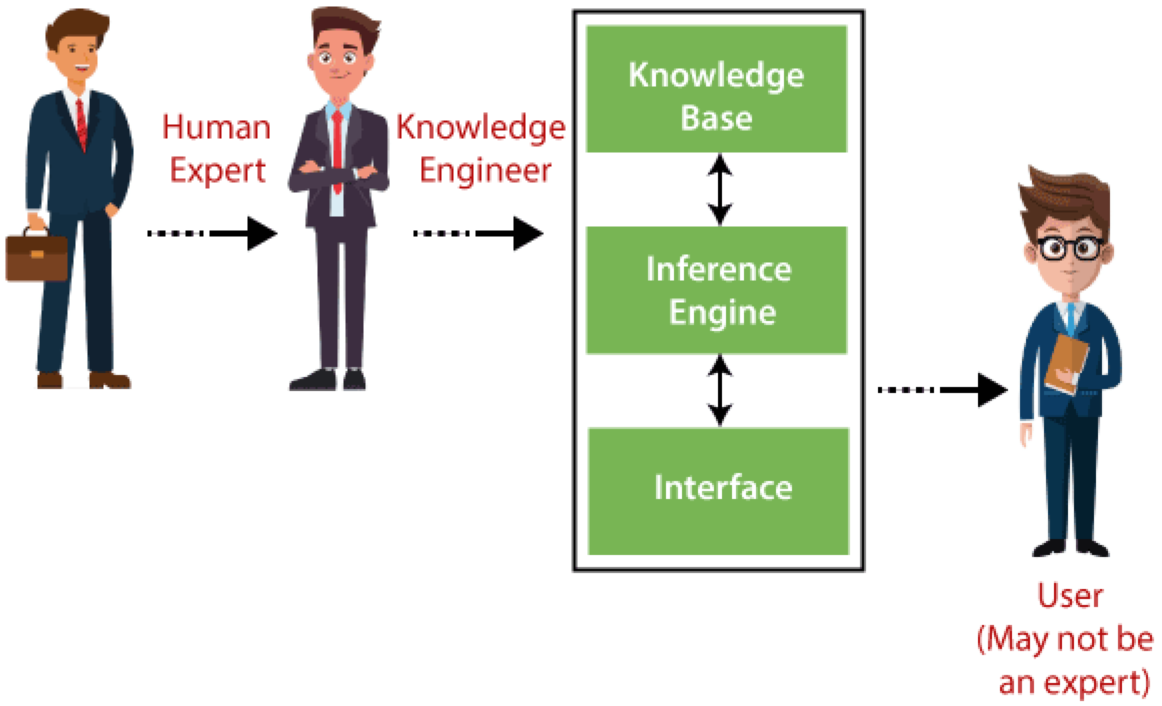
Characteristics of Expert System

- **High Performance:** The expert system provides high performance for solving any type of complex problem of a specific domain with high efficiency and accuracy.
- **Understandable:** It responds in a way that can be easily understandable by the user. It can take input in human language and provides the output in the same way.
- **Reliable:** It is much reliable for generating an efficient and accurate output.
- **Highly responsive:** ES provides the result for any complex query within a very short period of time.

Components of Expert System

An expert system mainly consists of three components:

- **User Interface** ○ **Inference Engine** ○ **Knowledge Base**



1. User Interface

With the help of a user interface, the expert system interacts with the user, takes queries as an input in a readable format, and passes it to the inference engine. After getting the response from the inference engine, it displays the output to the user. In other words, **it is an interface that helps a non-expert user to communicate with the expert system to find a solution**.

2. Inference Engine(Rules of Engine)
   - The inference engine is known as the brain of the expert system as it is the main processing unit of the system. It applies inference rules to the knowledge base to derive a conclusion or deduce new information. It helps in deriving an error-free solution of queries asked by the user.
   - With the help of an inference engine, the system extracts the knowledge from the knowledge base.
   - There are two types of inference engine:
   - **Deterministic Inference engine:** The conclusions drawn from this type of inference engine are assumed to be true. It is based on **facts** and **rules**.
   - **Probabilistic Inference engine:** This type of inference engine contains uncertainty in conclusions, and based on the probability.

Inference engine uses the below modes to derive the solutions:

- **Forward Chaining:** It starts from the known facts and rules, and applies the inference rules to add their conclusion to the known facts.
- **Backward Chaining:** It is a backward reasoning method that starts from the goal and works backward to prove the known facts.

3. Knowledge Base
    - The knowledgebase is a type of storage that stores knowledge acquired from the different experts of the particular domain. It is considered as big storage of knowledge. The more the knowledge base, the more precise will be the Expert System.
    - It is similar to a database that contains information and rules of a particular domain or subject.
    - One can also view the knowledge base as collections of objects and their attributes. Such as a Lion is an object and its attributes are it is a mammal, it is not a domestic animal, etc.

Components of Knowledge Base

- **Factual Knowledge:** The knowledge which is based on facts and accepted by knowledge engineers comes under factual knowledge.
- **Heuristic Knowledge:** This knowledge is based on practice, the ability to guess, evaluation, and experiences.

Knowledge Representation: It is used to formalize the knowledge stored in the knowledge base using the If-else rules.

Knowledge Acquisitions: It is the process of extracting, organizing, and structuring the domain knowledge, specifying the rules to acquire the knowledge from various experts, and store that knowledge into the knowledge base.

<u>Development of Expert System</u>

Here, we will explain the working of an expert system by taking an example of MYCIN ES. Below are some steps to build an MYCIN:

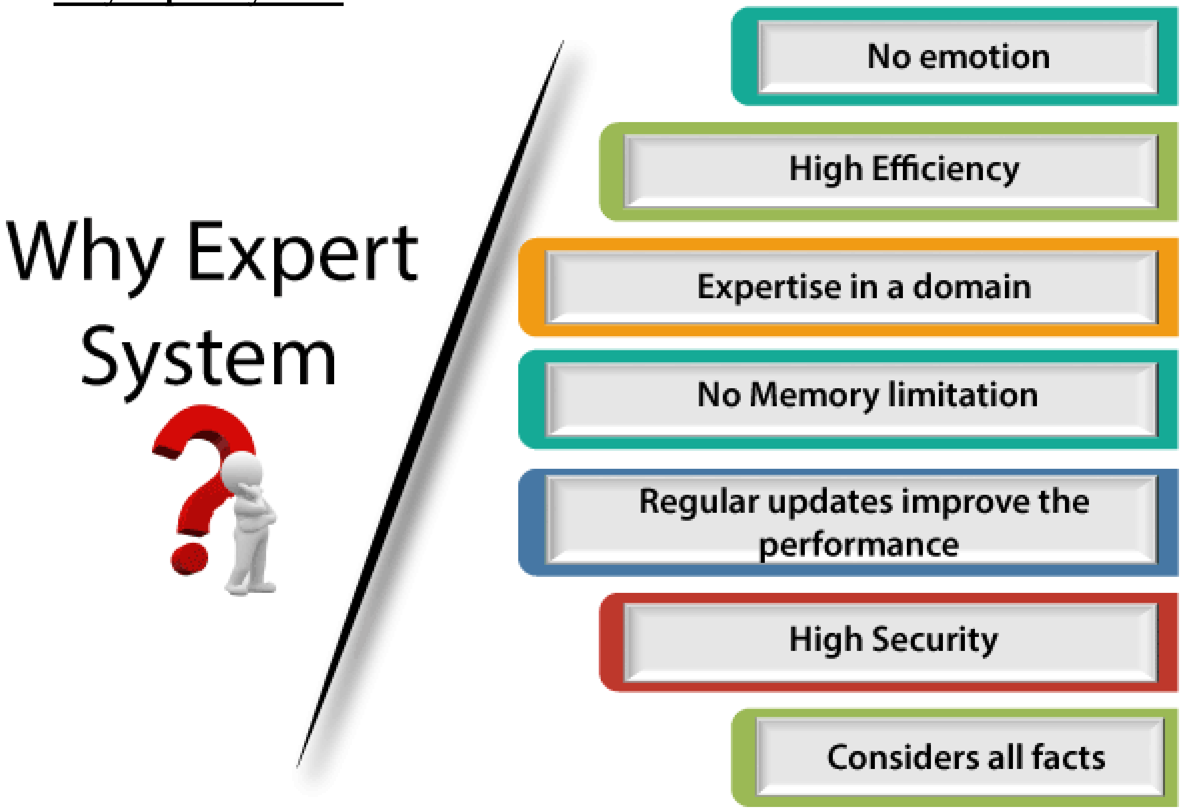
- Firstly, ES should be fed with expert knowledge. In the case of MYCIN, human experts specialized in the medical field of bacterial infection, provide information about the causes, symptoms, and other knowledge in that domain.
- The KB of the MYCIN is updated successfully. In order to test it, the doctor provides a new problem to it. The problem is to identify the presence of the bacteria by inputting the details of a patient, including the symptoms, current condition, and medical history.
- The ES will need a questionnaire to be filled by the patient to know the general information about the patient, such as gender, age, etc.
- Now the system has collected all the information, so it will find the solution for the problem by applying if-then rules using the inference engine and using the facts stored within the KB.
- In the end, it will provide a response to the patient by using the user interface.

<u>Participants in the development of Expert System</u>

There are three primary participants in the building of Expert System:

1. **Expert:** The success of an ES much depends on the knowledge provided by human experts. These experts are those persons who are specialized in that specific domain.
2. **Knowledge Engineer:** Knowledge engineer is the person who gathers the knowledge from the domain experts and then codifies that knowledge to the system according to the formalism.
3. **End-User:** This is a particular person or a group of people who may not be experts, and working on the expert system needs the solution or advice for his queries, which are complex.

Why Expert System?



Before using any technology, we must have an idea about why to use that technology and hence the same for the ES. Although we have human experts in every field, then what is the need to develop a computer-based system. So below are the points that are describing the need of the ES:

1. **No memory Limitations:** It can store as much data as required and can memorize it at the time of its application. But for human experts, there are some limitations to memorize all things at every time.
2. **High Efficiency:** If the knowledge base is updated with the correct knowledge, then it provides a highly efficient output, which may not be possible for a human.
3. **Expertise in a domain:** There are lots of human experts in each domain, and they all have different skills, different experiences, and different skills, so it is not easy to get a final output for the query. But if we put the knowledge gained from human experts into the expert system, then it provides an efficient output by mixing all the facts and knowledge
4. **Not affected by emotions:** These systems are not affected by human emotions such as fatigue, anger, depression, anxiety, etc.. Hence the performance remains constant.
5. **High security:** These systems provide high security to resolve any query.
6. **Considers all the facts:** To respond to any query, it checks and considers all the available facts and provides the result accordingly. But it is possible that a human expert may not consider some facts due to any reason.
7. **Regular updates improve the performance:** If there is an issue in the result provided by the expert systems, we can improve the performance of the system by updating the knowledge base.

Capabilities of the Expert System

- **Advising:** It is capable of advising the human being for the query of any domain from the particular ES.
- **Provide decision-making capabilities:** It provides the capability of decision making in any domain, such as for making any financial decision, decisions in medical science, etc.
- **Demonstrate a device:** It is capable of demonstrating any new products such as its features, specifications, how to use that product, etc.
- **Problem-solving:** It has problem-solving capabilities.
- **Explaining a problem:** It is also capable of providing a detailed description of an input problem.
- **Interpreting the input:** It is capable of interpreting the input given by the user.
- **Predicting results:** It can be used for the prediction of a result.
- **Diagnosis:** An ES designed for the medical field is capable of diagnosing a disease without using multiple components as it already contains various inbuilt medical tools.

Advantages of Expert System

- These systems are highly reproducible.
- They can be used for risky places where the human presence is not safe.
- Error possibilities are less if the KB contains correct knowledge.
- The performance of these systems remains steady as it is not affected by emotions, tension, or fatigue.
- They provide a very high speed to respond to a particular query.

Limitations of Expert System

- The response of the expert system may get wrong if the knowledge base contains the wrong information.
- Like a human being, it cannot produce a creative output for different scenarios.
- Its maintenance and development costs are very high. o Knowledge acquisition for designing is much difficult. o For each domain, we require a specific ES, which is one of the big limitations.
- It cannot learn from itself and hence requires manual updates.

Applications of Expert System

- **In designing &manufacturing domain**
  It can be broadly used for designing and manufacturing physical devices such as camera lenses and automobiles.
- **In the knowledge domain**
  These systems are primarily used for publishing the relevant knowledge to the users. The two popular ES used for this domain is an advisor and a tax advisor.
- **In the finance domain**
  In the finance industries, it is used to detect any type of possible fraud, suspicious activity, and advise bankers that if they should provide loans for business or not.
- **In the diagnosis and troubleshooting of devices**
  In medical diagnosis, the ES system is used, and it was the first area where these systems were used.
- **Planning and Scheduling**
  The expert systems can also be used for planning and scheduling some particular tasks for achieving the goal of that task.

Knowledge Acquisition

Knowledge acquisition refers to the process of extracting, structuring, and organizing domain knowledge from domain experts into a program. A **knowledge engineer** is an expert in AI language and knowledge representation who investigates a particular problem domain, determines important concepts, and creates correct and efficient representations of the objects and relations in the domain. Capturing domain knowledge of a problem domain is the first step in building an expert system. In general, the knowledge acquisition process through a knowledge engineer can be divided into four phases:

1. **Planning:** The goal is to understand the problem domain, identify domain experts, analyze various knowledge acquisition techniques, and design proper procedures.
2. **Knowledge extraction:** The goal is to extract knowledge from experts by applying various knowledge acquisition techniques.
3. **Knowledge analysis:** The outputs from the knowledge extraction phase, such as concepts and heuristics, are analyzed and represented in formal forms, including heuristic rules, frames, objects and relations, semantic networks, classification schemes, neural networks, and fuzzy logic sets. These representations are used in implementing a prototype expert system.
4. **Knowledge verification:** The prototype expert system containing the formal representation of the heuristics and concepts is verified by the experts. If the knowledge base is incomplete or insufficient to solve the problem, alternative knowledge acquisition techniques may be applied, and additional knowledge acquisition process may be conducted.

Many knowledge acquisition techniques and tools have been developed with various strengths and limitations. Commonly used techniques include interviewing, protocol analysis, repertory grid analysis, and observation.

Interviewing is a technique used for eliciting knowledge from domain experts and design requirements. The basic form involves free-form or unstructured question–answer sessions between the domain expert and the knowledge engineer. The major problem of this approach results from the inability of domain experts to explicitly describe their reasoning process and the biases involved in human reasoning. A more effective form of interviewing is called **structured interviewing,** which is goal-oriented and directed by a series of clearly stated goals. Here, experts either fill out a set of carefully designed questionnaire cards or answer questions carefully designed based on an established domain model of the problem-solving process. This technique reduces the interpretation problem inherent in the unstructured interviewing as well as the distortion caused by domain expert subjectivity.

As an example, let's look at the interviewing process used in constructing GTE's COMPASS system (Prerau, 1990). COMPASS is an expert system that examines error messages derived from a telephone switch's self-test routines and suggests running of additional tests or replacing a particular component.
The interviewing process in building COMPASS has an elicit–document–test cycle as follows:

1.
Elicit knowledge from an expert.
2.Document the elicited knowledge in rules and procedures.
3.Test the new knowledge using a set of data:
(a)Have the expert analyze a new set of data.

(b)Analyze the same set of data using the documented knowledge.

(c)Compare the two results.

(d) If the results differ, find the rules or procedures that lead to the discrepancy and return to step 1 to elicit more knowledge to resolve the problem.

Protocol analysis is another technique of data analysis originated in clinical psychology. In this approach, an expert is asked to talk about his or her thinking process while solving a given problem. The difference from interviewing is that experts find it much easier to talk about specific problem instances than to talk in abstract terms. The problem-solving process being described is then analyzed to produce a structured model of the expert's knowledge, including objects of significance, important attributes of the objects, relationships among the objects, and inferences drawn from the relationships. The advantage of protocol analysis is the accurate description of the specific actions and rationales as the expert solves the problem.
Repertory grid analysis investigates the expert's mental model of the problem domain. First, the expert is asked to identify the objects in the problem domain and the traits that differentiate them. Then, a rating grid is formed by rating the objects according to the traits.

Observation involves observing how an expert solves a problem. It enables the expert to continuously work on a problem without being interrupted while the knowledge is obtained. A major limitation of this technique is that the underlying reasoning process of an expert may not be revealed in his or her actions.

Knowledge acquisition is a difficult and time-consuming task that often becomes the bottleneck in expert system development (Hayes-Roth *et al*., 1983). Various techniques have been developed to automate the process by using domaintailored environments containing well-defined domain knowledge and specific problem-solving methods (Rothenfluh *et al*., 1996). For example, OPAL is a program that expedites knowledge elicitation for the expert system ONCOCIN (Shortliffe *et al*., 1981) that constructs treatment plans for cancer patients. OPAL uses a model of the cancer domain to acquire knowledge directly from an expert. OPAL's domain model has four main aspects:

entities and relationships, domain actions, domain predicate, and **procedural knowledge.**

Based on its domain knowledge, OPAL can acquire more knowledge from a human expert and translate it into executable code, such as production rules and finite state tables. Following OPAL, more general-purpose systems called PROTEGE and PROTEGE-II were developed (Musen, 1989). PROTEGE-II contains tools for creating domain ontology and generating OPAL-like knowledge acquisition programs for particular applications PROTEGE-II is a general tool developed by abstraction from a successful application, similar to the process from MYCIN to EMYCIN. Another example of automated knowledge acquisition is the SALT system (Marcus and McDermott, 1989) associated with an expert system called Vertical Transportation (VT) for designing customdesign elevator systems. SALT assumes a propose-and-revise strategy in the knowledge acquisition process. Domain knowledge is seen as performing one of three roles: (1) proposing an extension to the current design, (2) identifying constraints upon design extension, and (3) repairing constraint violation. SALT automatically acquires these kinds of knowledge by interacting with an expert and then compiles the knowledge into production rules to generate a domain-specific knowledge base. SALT retains the original knowledge in a declarative form as a dependency network, which can be updated and recompiled as necessary.

To build a knowledge base, the knowledge can be either captured through knowledge engineers or be generated automatically by machine learning techniques. For example, rules in rule-based expert systems may be obtained through a knowledge acquisition process involving domain experts and knowledge engineers or may be generated automatically from examples using decision-tree learning algorithms. Casebased reasoning is another example of automated knowledge extraction in which the expert system searches its collection of past cases, finds the ones that are similar to the new problem, and applies the corresponding solutions to the new one. The whole process is fully automatic. An expert system of this type can be built quickly and maintained easily by adding and deleting cases. Automatic knowledge generation is especially good when a large set of examples exist or when no domain expert exists.

In addition to generating knowledge automatically, machine learning methods have also been used to improve the performance of the inference engines by learning the importance of individual rules and better control in reasoning.